(2)

SHORT TERM TASK
SOFTWARE REQUIREMENTS SPECIFICATION and
SOFTWARE USERS MANUAL

CUT ORDER PLANNING

Co-Principal Investigator: Dr. Jane C. Ammons
Co-Principal Investigator: Dr. Charlotte Jacobs-Blecha
Research Investigator: Terri Smith
Research Assistant: Avril Baker
Research Assistant: Bill Warden

Prepared by: Terri Smith

Georgia Institute of Technology
Apparel Manufacturing Technology Center
Computer Science and Information Technology Laboratory
School of Industrial and Systems Engineering

October, 1989 - May, 1991

Research Sponsored by:
U.S. DEFENSE LOGISTICS AGENCY
Contract: DLA900-87-D-0018
Document Control No.: A-8496

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1991	Final Oct 1989-May 1991

4. TITLE AND SUBTITLE

Cut Order Planning

Short Term Task
Software Requirements Specification + Software users manual

5. FUNDING NUMBERS

DLA900-87-D-0018/0012

6. AUTHOR(S)

Jane Ammons and Charlotte Jacobs-Blecha

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Georgia Tech Research Institute
215 O'Keefe Building
Atlanta, GA 30332

8. PERFORMING ORGANIZATION REPORT NUMBER

A-8496

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Logistics Agency
Cameron Station (DLA-PRM)
Alexandria, VA 22304-6100

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified/Unlimited

12b. DISTRIBUTION CODE

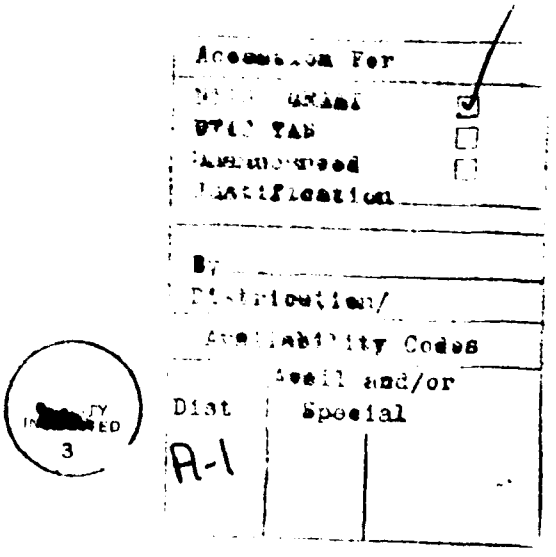
13. ABSTRACT (Maximum 200 words)

This report details the results of a research project conducted at the Georgia Institute of Technology which investigated methods for improving cut order planning in apparel manufacturing. The project had two complementary objectives. The first objective was to investigate existing solution methodologies for the cut order planning problem. Alternate commercial software packages were examined and their performances comparatively analyzed, using testbed data representative of industrial problems.

The results of this research provide important insights into the state-of-the-art in COP solution methods. A mathematical model of the COP problem was developed to facilitate problem specification and to initiate heuristic development. As a result of the complexity analysis, the COP problem was shown to be sufficiently complex that heuristic methods are the only reasonable means of finding solutions in real time. New methods were developed which perfrm as well as or better than those used in existing commercial packages. These algorithms have been implemented in a prototype software package for easy incorporation into existing commercial software, and will be transferred to industry through a participating s/w vendor.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Production Planning, Cut Order Planning Software, Apparel Manufacturing.	322
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED



1.0 Scope

1.1 Identification

This User's Manual provides information, to users and programmers, about three algorithms for the Cut Order Planning problem: Savings, Cherry Picking, and Improvement. This information will enable the user to operate the programs and give the software programmer a better understanding of the software written.

1.2 Purpose

The algorithms were designed to analyze the cut order planning problem. The first algorithm uses a "computed savings" to combine sections in a cut plan. The second algorithm is also a constructive algorithm which combines pieces into one section at a time. The third algorithm tries to improve an existing solution by replacing pieces into other sections. The algorithms presented in this document are used for testing purposes only.

2.0 Equipment Configuration

The software was written on an IBM PC platform with MSDOS environment. All code is written in the Ansi C format. The outputs were generated with a 386/25MHz machine.

3.0 User Requirements

A typical user of the algorithms should be familiar with basic computer skills. The user needs to know how to edit an Ascii file (input file) and print a listing of the output file either on the screen or to a printer.

A software programmer must have knowledge of the C programming language to edit any of the procedures, and should have knowledge of either the Pascal or C languages to read the code. The code contains basic array structures, but no linked list structures. Therefore the user does not need extensive knowledge with the use of pointers but needs to understand arrays and the allocation of memory for such array structures. The software programmer must also be familiar with makefiles and how to compile and link the separate procedures.

4.0 Algorithm Description

4.1 Savings Algorithm

4.1.1 Functional Description

The Savings algorithm uses a "computed savings" to combine sections in a cut plan. The algorithm begins with each unit of fabric ordered in a separate section of initial ply height. Sections in this list are continuously merged into one section based on which merges provide the best savings in inches of fabric. As two sections are merged the new section is placed in a partial section. This partial section is then merged with other sections until the maximum sizes allowed per section is reached. Once this partial section is full, it is saved in a permanent list of sections and can no longer be used to merge with other sections. At this point a new partial section is created by combining two sections and the process begins again. Sections are only merged if the new section created does not exceed the maximum ply height or the maximum sizes allowed per section. When no more mergers are possible the solution is written to an output file and the program terminates.

Before merging sections together, the program calculates a potential savings in fabric for each possible combination of sections. There are basically two ways to merge two sections. If the two sections contain the same size combination, the savings can be computed by placing one section on top of another. The savings for this method is based on the saved cutting cost, since the size combination is only cut once instead of twice. The merger for sections containing the same size combination can also be achieved by changing the size combination to include two of each of the original sizes and leaving the ply height the same. The savings for this method is the decreased cost of fabric required for spreading the merged sections. If the two sections do not contain the same sizes then the savings is computed by changing the size combination and leaving the ply height the same.

Appendix C describes the Savings algorithm in detail and Appendix D contains a printout of the source code for the Savings algorithm.

The Savings algorithm is run simply by typing in "savings" at the DOS prompt in the directory in which the executable program is located. An input file, explained in section 4.1.2, must reside in the current directory in which the user is running the Savings algorithm. All output is written to a file also in that current directory.

4.1.2 Input File

The input file is an Ascii file which must contain the variables needed to run the Savings algorithm. The inputs include (1) an order to be cut, consisting of various sizes required, the quantity desired of each of these sizes, and

the perimeter around the size to cut. (2) The number of units over or under the demand that will be allowed. (3) The parameter K which determines the number of iterations after which the savings list will be updated. (4) The ply height of each of the initial sections. (5) List of L_i's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section. (6) The maximum ply height allowed. (7) The maximum number of sizes allowed per section. (8) The cutting cost per inch of fabric. (9) The unit cost of the fabric.

Each input is placed on a separate line on the input file and must be placed in the following order represented below:

```
int ou_units        : number of units over or under
                      the demand
int max_ply         : maximum ply height
int max_sizes       : max sizes allowed per section
int init_ply        : initial ply height
int k               : number of iterations after which
                      the savings list is updated
int q               : the ply height in which to use
int cut_cost        : cutting cost per inch
int unit_cost       : unit cost of size
order_t order       : the order
list_t list         : list of Li's (fabric lengths
                      for each size combination)
```

The beginning of the input file would appear as follows:

```
3                : ou_units
3                : max_sizes
4                : max_ply
1                : init_ply
1                : k
1                : cut_cost
1                : unit_cost
5 120 size-30    : order ( #, perimeter, string for size)
5 130 size-32    : order ( #, perimeter, string for size)
5 140 size-34    : order ( #, perimeter, string for size)
1 0 -1 15.31     : list  ( 1 size 0, inches)
1 1 -1 15.56     : list  ( 1 size 1, inches)
.
.
.
-2               : end
```

A detailed description of the data structures for an order and the list of L_is is in section 4.1.4.

As each row of the order is read from the input file the values are placed in the ord_var_t structure. For example the first row (5 120 size-30) is placed as follows:

```
number[0] = 5
ch_sizes[0] = "size-30"
perimeter[0] = 120
```

As each row in the list of L_is is read the variable are placed in the list_t array as follows:

```
List[0].sizes[0] = 1    ( 1 size-30 )
List[0].sizes[1] = 0    ( 0 size-32 )
List[0].inches = 15.31

List[1].sizes[0] = 0    ( 0 size-30 )
List[1].sizes[1] = 1    ( 1 size-32 )
List[1].inches = 15.56
```

Appendix A contains a printout of the entire input file.

4.1.3 Output File

The output file ("OUTPUT") lists the sizes in each section, the inches for that section, the ply height, and the total inches for the section (ply x inches). At the end the total inches are printed along with the number of units over or under the demand. A sample output file is in Appendix B.

4.1.4 Data Structures

MAX_SIZES defines the maximum number of different sizes possible in one order.

MAX_LIST defines the maximum number of possible size combinations as in the input file.

MAX_SAVINGS defines the maximum number of savings that will be kept in a list.

The following structure holds the order which consists of the amount of each size needed, the string value that will be written to the output file associated with that size, and the perimeter of the particular size:

```
struct order {
   order_t number;
   sizes_t ch_sizes;
   int     perimeter[MAX_SIZES];
   } ord_var_t;
```

Order.number is an array which holds the amount of each size ordered. It is defined as follows: typedef int order_t[MAX_SIZES].

Order.ch_sizes is an array which holds the string value associated with the size. e.g "Size-32". It is

defined as follows:
typedef char sizes_t[MAX_SIZES][10] :

Order.perimeter is an array which holds the perimeter of each size.

Arrays in the C language begin with zero and not one. Therefore there will be a value in the first position, zero. If the order is 3 size-30 and 5 size-32, the ord_var_t structure will hold the following numbers:

```
number[0] = 3              number[1] = 5
ch_sizes[0] = "size-30"    ch_sizes[1] = "size-32"
perimeter[0] = 120         perimeter[1] = 130
```

The following structure holds the list of size combinations:

```
struct list_t {
   order_t sizes;
   float   inches:
   } list
```

List.sizes is an array which holds the amount of each size in the specific size combination.

List.inches holds the amount of fabric in inches need for that specific sizes combination.

An example of what is in the list[] structure:

```
sizes[0] = 1   ( 1 size-30 )
sizes[1] = 1   ( 1 size-32)
inches = 28.01
```

The following structure holds the data necessary for each section.

```
struct section_t {
   order_t sizes;
   int     ply_height;
   char    merged;
   } section
```

Section.order is an array which holds the amount of each size in the specific size combination for this particular section.

Section.ply_height is the ply height of that section.

Section.merged tells whether the section has already been merged or not.

The following structure contains the data for each savings calculated between two sections:

```
struct savings_t {
   int    sect1
   int    sect2
   int    ply_height;
   float  savings;
   int    type;
   int    ply1;
   int    ply2;
   } save_list;
```

Save_list.sect1 and save_list.sect2 are the numbers of the two sections being merged. These numbers correspond to their position in the section list.

Save_list.ply_height is the ply height of the new combined section.

Save_list.savings is the savings achieved if these two sections are merged.

Save_list.type is the type of merge that is needed.
   1 : Sizes the same, place on top of another
   2 : Sizes different, rearrange size combination

Save_list.ply1 and save_list.ply2 are the different ply heights of sect1 and sect2.

4.1.5 Module Definitions

4.1.5.1 Case_ai.c

Case_ai() computes the savings of placing one section on top of the other. The two sections must have the same size combination.

4.1.5.2 Case_aii.c

Case_aii() computes the savings of rearranging the size combination of the two sections into one section but keeping the ply height the same.

4.1.5.3 Compute.c

Compute_savings() looks at the two sections to determine which way to compute the savings. If the two sections have the same sizes combination then case_ai.c is called else case_aii.c is called. Ply height of the sections are adjusted and the savings is returned to the main program.

4.1.5.4 Findinch.c

Find_inches() searches through the list of L_i inputs to find the inches for the specific size combination of a section.

4.1.5.5 Getparm.c

Get_parameters() opens the input file and reads in all the inputs into their appropriate variables and structures.

4.1.5.6 Globals.c

Globals defines and initializes all the global variables for the Savings algorithm.

4.1.5.7 Savings.c

This file contains the main program to execute the savings algorithm. The procedure begins by allocating enough memory for all the variables and lists and then calls get_parameters() to read the input file. The main loop in the procedure calculates the savings and merges sections until no more merges are possible by calling the other various procedures. This procedure then writes all the final results to the file "Output". All allocated space is freed and all files are closed.

4.1.5.8 Savdec.h

This file contains all the definitions for various structures and procedures that are used by the Savings algorithm.

4.1.5.9 Savelcl.h

This file defines the global variables for the Savings algorithm as external variables so that the program will compile. The actual definition and initialization is in the globals.c file.

4.1.5.10 Makefile.pc

This file contains the method in which to compile all the procedures and create the executable file Savings.exe. To execute this file type "make makefile.pc".

4.1.6 Error Messages

All error messages provide a short message and the file in which the error occurred.

The following message appears when the program has trouble opening a file in which to write the output.

"CANNOT OPEN OUTPUT FILE savings.c"

The following message appears when the program has trouble allocating memory to any structure needed for the program.

"ALLOCATION ERROR for list savings.c"

The following error appears when the program tries a specific combination of sizes in one section but can not locate the inches for this combination in the input list. The user should check the input file to determine if the value exists or not.

"CANNOT FIND 1 size-30 2 size-32 findinch.c"

4.2 Cherry Picking Algorithm

4.2.1 Functional Description

The Cherry Picking algorithm builds sections by combining certain sizes based on the best utilization of fabric. The algorithm begins by choosing the first (Q1) and second (Q2) most numerous sizes in the order. OU will represent the number of units over or under the demand that is allowed. Any size which has an order quantity greater than (Q2 - OU) is placed into sections such that a minimal amount of fabric is used. All quantities of sizes assigned to a section are reduced appropriately. This process continues until all sizes have been assigned to sections. When this occurs the program terminates and the output is written to a file.

Appendix C describes the algorithm in greater detail and Appendix E contains the source code for the Cherry Picking algorithm.

The Cherry algorithm is run simply by typing in "cherry" at the DOS prompt in the directory in which the executable program is located. An input file, explained in section 4.2.2, must reside in the current directory in which the user is running the Cherry algorithm. All output is written to a file also in that current directory.

4.2.2 Input File

The input file is an Ascii file which must contain the variables needed to run the Savings algorithm. The inputs include (1) an order to be cut, consisting of various sizes required, and the quantity desired of each of these sizes. (2) The number of units over or under the demand that will be allowed. (3) List of L_i's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section. (4) The maximum ply height allowed. (5) The maximum number of sizes allowed per section.

Each input is placed on a separate line on the input file and must be placed in the following order represented below:

```
int ou_units        : number of units over or under
                      the demand
int max_ply         : maximum ply height
int max_sizes       : max sizes allowed per section
order_t order       : the amount of each size needed
list_t list         : list of I's (fabric lengths
                      for each sizes combination)
```

The input file would appear as follows:

```
3                 : ou_units
3                 : max_sizes
4                 : max_ply
5 120 size-30     : order ( #, perimeter, string for size)
5 130 size-32     : order ( #, perimeter, string for size)
5 140 size-34     : order ( #, perimeter, string for size)
1 0 -1 15.31      : list  ( 1 size 0, inches)
1 1 -1 15.56      : list  ( 1 size 1, inches)
.
.
.
-2                : end
```

A detailed description of the data structures for an order and the list of L_is is in section 4.2.4.

As each row of the order is read from the input file the values are placed in the ord_var_t structure. For example the first row (5 120 size-30) is placed as follows:

```
number[0] = 5
ch_sizes[0] = "size-30"
perimeter[0] = 120
```

As each row in the list of L_is is read the variable are placed in the list_t array as follows:

```
List[0].sizes[0] = 1     ( 1 size-30 )
List[0].sizes[1] = 0     ( 0 size-32 )
List[0].inches = 15.31

List[1].sizes[0] = 0     ( 0 size-30 )
List[1].sizes[1] = 1     ( 1 size-32 )
List[1].inches = 15.56
```

Appendix A contains a printout of the entire input file.

4.2.3 Output File

The output file ("OUTPUT") lists the sizes in each section, the inches for that section, the ply height, and the total inches for the section (ply x inches). At the end the total inches are printed along with the number of units over or under the demand. A sample output file is in Appendix B.

4.2.4 Data Structures

MAX_SIZES defines the maximum number of different sizes possible in one order.

MAX_LIST defines the maximum number of possible size combinations as in the input file.

The following structure holds the order which consists of the quantity amount of each size needed, the string value that will be written to the output file associated with that size, and the perimeter of the particular size:

```
struct order {
   order_t number;
   sizes_t ch_sizes;
   int     perimeter[MAX_SIZES];
   } ord_var_t;
```

Order.number is an array which holds the amount of each size ordered. It is defined as follows:
typedef int order_t[MAX_SIZES].

Order.ch_sizes is an array which holds the string value associated with the size. e.g "Size-32". It is defined as follows:
typedef char sizes_t[MAX_SIZES][10] :

Order.perimeter is an array which holds the perimeter of each size.

Arrays in the C language begin with zero and not one. Therefore there will be a value in the first position, zero. If the order is 3 size-30 and 5 size-32, the ord_var_t structure will hold the following numbers:

```
number[0] = 3              number[1] = 5
ch_sizes[0] = "size-30"    ch_sizes[1] = "size-32"
perimeter[0] = 120         perimeter[1] = 130
```

The following structure holds the list of L_is:

```
struct list_t {
   order_t sizes;
   float   inches:
   } list
```

List.sizes is an array which holds the amount of each size in the specific size combination.

List.inches holds the amount of fabric in inches need for that specific sizes combination.

An example of what is in the list[] structure:

```
sizes[0] = 1   ( 1 size-30 )
sizes[1] = 1   ( 1 size-32)
inches = 28.01
```

The following structure holds the data necessary for each section.

```
struct section_t {
   order_t sizes;
   int     ply_height;
   } section
```

Section.order is an array which holds the amount of each size in the specific size combination for this particular section.

Section.ply_height is the ply height of that section.

4.2.5 Module Descriptions

4.2.5.1 Cherry.c

This file contains the main program which executes the Cherry Picking algorithm. The procedure begins by allocating memory for all the variables and lists for the program and calls get_parameters() to read the input file. The main loop in the program chooses the best combination of sizes in the set S by calling the other various procedures. After all sizes have been used the procedure writes the results to the file "Output", releases all memory, and closes all files.

4.2.5.2 Chkinch.c

Check_inches() detremines if the total inches calculated from the current combination is less than the previous best combination. If so this new combination is saved in a temporary section.

4.2.5.3 Clrtemp.c

Clear_temp() initializes the temporary sections.

4.2.5.4 Combine.c

Combine_inches calls find_inches() to calculated the total inches for a specific combination of sizes.

4.2.5.5 Cphold.c

Copy_hold_to_sections() copies all the sections in the temporary holding segment to the final output hold segment of sections.

4.2.5.6 Findinch.c

Find_inches searches through the list of L_i inputs to find the inches for a specific size combination.

4.2.5.7 Fives.c

Fives() is a recursive procedure which groups sizes in combinations of five. Recursive programming means it keeps calling itself until all possible combinations are exhausted.

4.2.5.8 Fours.c

Fours() is a recursive procedure which groups sizes in combinations of four. Recursive programming means it keeps calling itself until all possible combinations are exhausted.

4.2.5.9 Getparm.c

Get_parameters() reads the input file and places all input variables in their appropriate structures.

4.2.5.10 Globals.c

Globals.c defines and initializes all global variables for the Cherry Picking algorithm.

4.2.5.11 Ones.c

Ones() places each sizes in a section by itself and find the total inches for all these sections combined.

4.2.5.12 Sixes.c

Sixes() is a recursive procedure which groups sizes in combinations of six. Recursive programming means it keeps calling itself until all possible combinations are exhausted.

4.2.5.13 Threes.c

Threes() is a recursive procedure which groups sizes in combinations of three. Recursive programming means it keeps

calling itself until all possible combinations are exhausted.

4.2.5.14 Twos.c

Twos() is a recursive procedure which groups sizes in combinations of two. Recursive programming means it keeps calling itself until all possible combinations are exhausted.

4.2.5.15 Cherdec.h

This file contains all the definitions for various structures and procedures that are used by the Cherry Picking algorithm.

4.2.5.16 Cherlcl.h

This file defines the global variables for the Cherry Picking algorithm as external variables so that the program will compile. The actual definition and initialization is in the globals.c file.

4.2.5.17 Makefile.pc

This file contains the method in which to compile all the procedures and create the executable file Cherry.exe. To execute this file type "make makefile.pc".

4.2.6 Error Messages

All error messages give a short message and the file in which the error occurred.

The following message appears when the program has trouble opening a file in which to write the output.

"CANNOT OPEN OUTPUT FILE cherry.c"

The following message appears when the program has trouble allocating memory to any structure needed for the program.

"ALLOCATION ERROR for list cherry.c"

The following error appears when the program tries a specific combination of sizes in one section but can not locate the inches for this combination in the input list. The user should check the input file to determine if the value exists or not.

"CANNOT FIND 1 size-30 2 size-32 findinch.c"

4.3 Improvement Algorithm

4.3.1 Functional Description

4.3.1.1 Improvement Algorithm on current solution

The Improvement algorithm takes a current solution for a cut plan and tries to improve this solution by exchanging sizes in different sections. The algorithm begins by first examining the current solution to see if any sections can be combined to make one section that requires less fabric than the two sections. The new section cannot violate the constraints on the maximum ply height or the maximum sizes allowed per section. Then the algorithm begins with any section and tries to transfer a portion of the section to another section, or swap a portion of the section with the portion of another section without violating the constraint

of the maximum number of sizes allowed per section. When all possible transfers and swaps have been examined for the current section, the best transfer or swap is made and the process begins again for the next section. This process is continued until no more transfers or swaps that improve the solution can be made. At this point the new solution is written to the output file and the program terminates.

Before actually transferring or swapping a portion of a section, the program calculates a possible savings in fabric for each possible combination of sections. There are basically two ways to transfer or swap sizes. If the portion taken from one section and the candidate section contain the same size combination the savings is computed by placing the portion on top of the candidate section. The savings for this method is based on the saved cutting cost since the size combination is only cut once instead of twice. The savings for the merger of a portion of the original section and the entire candidate section containing the same size combination can also be achieved by adding the sizes in the extracted portion to the new section and leaving the ply height the same. The savings for this method is the decreased cost of fabric required for spreading the merged sections. If the portion taken from one section and candidate section do not contain the same size combination then the savings is computed by changing the size combination and leaving the ply height the same.

4.3.1.2 Improvement Algorithm on an order

The Improvement algorithm can also be used to generate a solution from an initial order by either changing the source code slightly or the input file. To accomplish this each unit in the initial order must be placed in a separate section of initial ply height. The improvement algorithm

works as if this were the initial solution and tries to improve upon this solution as explained above.

There are two ways to set up this initial solution for the algorithm. The user can edit the input file to show a initial solution of sections with initial ply heights and one unit of clothing in each section. (See section 4.3.2 Input File). However if the order is large it would be easier to change the source code to omit reading in the first solution from the input file and simply place each size ordered into a section of initial ply height. The two files which would need to be changed are the getparm.c file and the improve.c file. In the getparm.c file the code that reads in the initial solution would have to be deleted. In the improve.c file new source code would have to be added which would take the order and place each unit of fabric into a separate section of initial ply height. This code is represented below:

```
curr_section = 0;
for (i=0; i< num_of_sizes; i++) {

   for (j=0; j<order[i]; j++) {
      sections[curr_section].ply_height =
                                    initial_ply;
      sections[curr_section].sizes[i] = 1;
      ++curr_section;
      }
   }
```

The Improvement algorithm will then generate a solution based on the order and write out the solution to the output file.

Appendix C contains a detailed description of the algorithm and Appendix F contains the source code for the Improvement algorithm.

The Improvement algorithm is run simply by typing in "improve" at the DOS prompt in the directory in which the Improvement program is located. An input file, explained in section 4.3.2 needs to be in the current directory in which the user is running the Improvement algorithm. All output is written to a file also in that current directory.

4.3.2 Input File

The input file is an Ascii file which must contain the variables needed to run the Improvement algorithm. The inputs include (1) an order to be cut, consisting of various sizes required, and the quantity desired of each of these sizes (2) The number of units over or under the demand that will be allowed. (3) List of L_i's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section. (4) A solution to the problem to be improved upon. The solution contains the number of sections and the sizes and ply height assigned to each of those sections, the fabric length required for each of the sections, and the deviation of the number of units to be cut from the actual number of units required in the order. (5) The maximum ply height allowed. (6) The maximum number of sizes allowed per section. (7) The cutting cost per inch of fabric. (8) The unit cost of the fabric.

Each input is placed on a separate line on the input file and must be placed in the following order represented below:

```
int ou_units        : number of units over or under
                      the demand for 1st solution
int max_ply         : maximum ply height
```

```
int max_sizes          : max sizes allowed per section
int cut_cost           : cutting cost per inch
int unit_cost          : unit cost of size
order_t order          : the amount of each size needed
num_sections           : the # of sections in 1st solution
sections_t section     : the sections for the 1st solution
                         (sizes in section, ply height)
old_ou_units           : # of units over or under demand
                         for new solution
list_t list            : list of I's (fabric lengths
                         for each sizes combination)
```

The beginning of the input file would appear as follows:

```
3                  : ou_units
3                  : max_sizes
4                  : max_ply
1                  : init_ply
1                  : k
1                  : cut_cost
1                  : unit_cost
5 120 size-30      : order ( #, perimeter, string for size)
5 130 size-32      : order ( #, perimeter, string for size)
5 140 size-34      : order ( #, perimeter, string for size)
5 150 size-36      : order ( #, perimeter, string for size)
2                  : number of sections
1 0 2 3 -1 2       : sections (sizes , ply height)
1 2 2 3 -1 3       : sections (sizes , ply height)
3                  : ou_units for new_section
1 0 -1 15.31       : list  ( 1 size 0, inches)
1 1 -1 15.56       : list  ( 1 size 1, inches)
.
.
.
-2                 : end
```

A detailed description of the data structures for an order, the list of L_is, and a solution is in section 4.3.4.

As each row of the order is read from the input file the values are placed in the ord_var_t structure. For example the first row (5 120 size-30) is placed as follows:

```
number[0] = 5
ch_sizes[0] = "size-30"
perimeter[0] = 120
```

The input file for the original solution reads as follows:

```
1 size-30, 2 size-36 with a ply height of 2
1 size-34, 2 size-36 with a ply height of 3
```

As each row in the input file for the original solution is read the variables are placed in a section list as follows:

```
section[0].sizes[0] = 1
section[0].sizes[1] = 0
section[0].sizes[2] = 0
section[0].sizes[3] = 2
section[0].ply_height = 2

section[1].sizes[0] = 0
section[1].sizes[1] = 0
section[1].sizes[2] = 1
section[1].sizes[3] = 2
section[1].ply_height = 3
```

As each row in the list of L_is is read the variable are placed in the list_t array as follows:

```
List[0].sizes[0] = 1   ( 1 size-30 )
List[0].sizes[1] = 0
List[0].inches = 15.31

List[1].sizes[0] = 0
List[1].sizes[1] = 1
List[1].inches = 15.56
```

Appendix A contains a printout of the entire input file.

4.3.3 Output File

The output file ("OUTPUT") lists both the first solution, given as the input, the second solution generated by the algorithm, the sizes in each section, the inches for that section, the ply height, and the total inches for the section (ply x inches). At the end the total inches are printed along with the number of units over or under the demand. A sample output file is in Appendix B.

4.3.4 Data Structures

MAX_SIZES defines the maximum number of different sizes possible in one order.

MAX_LIST defines the maximum number of possible size combinations as in the input file.

The following structure holds the order which consists of the amount of each size needed, the string value that will be written to the output file associated with that size, and the perimeter of the particular size:

```
struct order {
   order_t number;
   sizes_t ch_sizes;
```

```
   int     perimeter[MAX_SIZES];
   } ord_var_t;
```

Order.number is an array which holds the amount of each size ordered. It is defined as follows:
typedef int order_t[MAX_SIZES].

Order.ch_sizes is an array which holds the string value associated with the size. e.g "Size-32". It is defined as follows:
typedef char sizes_t[MAX_SIZES][10] :

Order.perimeter is an array which holds the perimeter of each size.

Arrays in the C language begin with zero and not one. Therefore there will be a value in the first position, zero. If the order is 3 size-30 and 5 size-32, the ord_var_t structure will hold the following numbers:

```
number[0] = 3              number[1] = 5
ch_sizes[0] = "size-30"    ch_sizes[1] = "size-32"
perimeter[0] = 120         perimeter[1] = 130
```

The following structure holds the list of L_is:

```
struct list_t {
   order_t sizes;
   float   inches:
   } list
```

List.sizes is an array which holds the amount of each size in the specific size combination.

List.inches holds the amount of fabric in inches need for that specific sizes combination.

An example of what is in the list[] structure:

```
sizes[0] = 1   ( 1 size-30 )
sizes[1] = 1   ( 1 size-32)
inches = 28.01
```

The following structure holds the data necessary for each section.

```
struct section_t {
   order_t sizes;
   int     ply_height;
   } section
```

Section.order is an array which holds the amount of each size in the specific size combination for this particular section.

Section.ply_height is the ply height of that section.

The following structure contains the data for all the savings calculated and saved in a list:

```
struct savings_t {
   int    sect1
   int    sect2
   int    org_ply_height
   int    cand_ply_height
   float  savings;
   int    type;
   order_t org;
   order_t cand;
   order_t in_sect1;
   order_t in_sect2;
   } save_list;
```

Save_list.sect1 and save_list.sect2 are the numbers of the two sections being merged. These numbers correspond to their position in the section list.

Save_list.org_ply_height is the ply height of the originating section in which the portion was taken.

Save_list.cand_ply_height is the ply height of the candidate section in which the portion will be added or swapped.

Save_list.savings is the savings achieved if these two sections are merged.

Save_list.type is the type of merge that is needed.
   1 : Sizes the same, place on top of another
   2 : Sizes different, rearrange size combination

Save_list.org is an array which holds sizes of the originating portion to transfer or swap.

Save_list.cand is an array which holds the sizes of the candidate portion to transfer or swap.

Save_list.in_sect1 is an array which holds all the sizes in the original section before the swap or transfer is made.

Save_list.in_sect2 is an array which holds all the sizes in the candidate section before the swap or transfer is made.

4.3.5 Module Descriptions

4.3.5.1 Case_ai.c

Case_ai() computes the savings of placing one section or portion of a section on top of another section. The two sections must have the same size combination.

4.3.5.2 Case_aii.c

Case_aii() computes the savings of rearranging the size combination of the two sections into one section but keeping the ply height the same.

4.3.5.3 Combply.c

Combine_ply() combines sections which have the same sizes combination into one section if the new ply height does not violate the maximum ply allowed.

4.3.5.4 Combsize.c

Combine_sizes() combines sections which have the same ply height if the number of sizes in the section does not violate the maximum number of sizes allowed per section.

4.3.5.5 Compswap.c

Compute_swap_savings() looks at the two sizes and sections to swap to determine which way to compute the savings. If the two sections have the same size combination then case_ai.c is called, else case_aii.c is called. Ply height of the sections are adjusted and the savings is returned to the main program.

4.3.5.6 Compute.c

Compute_savings() looks at the size to transfer and the two sections to determine which way to compute the savings. If the two sections have the same sizes combination then case_ai.c is called else case_aii.c is called. Ply height of the sections are adjusted and the savings is returned to the main program.

4.3.5.7 Findinch.c

Find_inches() searches through the list of L_i inputs to find the inches of the size combination of a section.

4.3.5.8 Getparm.c

Get_parameters() opens the input file and reads in all the inputs into their appropriate variables and structures.

4.3.5.9 Globals.c

Globals defines and initializes all the global variables for the Improvement algorithm.

4.3.5.10 Improve.c

This file contains the procedure to execute the Improvement algorithm. The procedure begins by allocating all the memory needed for the variables and lists, and then calls get_parameters() to read the input file. The main loop in the procedure continually tries to transfer and swap

sizes between sections until no more transfers or swaps can be made to improve the solution. The procedure then writes all the final data to the file "Output", frees all memory, and closes any open files.

4.3.5.11 Swapbkwd

Swap_backwards() attempts to swap one size from one section with a size from another section working backwards through the list.

4.3.5.12 Swapfrwd

Swap_forwards() attempts to swap one size from one section with a size from another section working forwards through the list.

4.3.5.13 Tranbkwd

Transfer_backwards() attempts to transfer one size from one section to another section working backwards through the list.

4.3.5.14 Tranfrwd

Transfer_forwards() attempts to transfer one size from one section to another section working forwards through the list.

4.3.5.15 Impdec.h

This file contains all the definitions for various structures and procedures that are used by the Improvement algorithm.

4.3.5.16 Implcl.h

This file defines the global variables for the Improvement algorithm as external variables so that the

program will compile. The actual definition and initialization is in the globals.c file.

4.3.5.17 Makefile.pc

This file contains the method in which to compile all the procedures and create the executable file Improve.exe. To execute this file type "make makefile.pc".

4.3.6 Error Messages

All error messages give a short message and the file in which the error occurred.

The following message appears when the program has trouble opening a file in which to write the output.

"CANNOT OPEN OUTPUT FILE improve.c"

The following message appears when the program has trouble allocating memory to any structure needed for the program.

"ALLOCATION ERROR for list improve.c"

The following error appears when the program tries a specific combination of sizes in one section but can not locate the inches for this combination in the input list. The user should check the input file to determine if the value exists or not.

"CANNOT FIND 1 size-30 2 size-32 findinch.c"

5.0 References

"Cut Order Planning Final Report", May 1991. Co-project Directors: Dr. Jane C. Ammons, Dr. Charlotte Jacobs-Blecha.

"Microsoft C CodeView And Utilities, Software Development Tools For The MS-DOS Operating System", 1987. Microsoft Corporation.

"Microsoft C For The MS-DOS Operating System, Run-Time Library Reference", 1987. Microsoft Corporation.

"Microsoft C Optimizing Compiler For The MS-DOS Operating System, User's Guide, 5.0 with 5.1 Update", 1987. Microsoft Corporation.

6.0 Appendices

Appendix A Input File

Each algorithm has a unique input file. The L_is in the three input files (one for each algorithm) are all the same. However, each input file begins slightly different depending on the other input variables needed for the algorithm. All other input files are the same except for the initial inputs. The input file in this appendix is used in the Savings algorithm. The beginning of each of the three input files are explained in the sections 4.1.2 Savings Input File, 4.2.2 Cherry Input File, 4.3.2 Improvement Input File.

```
 1    0
 2    47
 3    6
 4    1
 5    1
 6    1
 7    1
 8    1
 9    72 120 size-30
10    144 130 size-32
11    360 140 size-34
12    360 150 size-36
13    144 160 size-38
14    72 170 size-40
15    -1
16    1 0 -1 15.31
17    1 1 -1 15.56
18    1 2 -1 15.82
19    1 3 -1 16.07
20    1 4 -1 16.33
21    1 5 -1 16.59
22    2 0 -1 27.75
23    2 1 -1 28.26
24    2 2 -1 28.76
25    2 3 -1 29.27
26    2 4 -1 29.77
27    2 5 -1 30.28
28    3 0 -1 40.36
29    3 1 -1 41.11
30    3 2 -1 41.87
31    3 3 -1 42.63
32    3 4 -1 43.38
33    3 5 -1 44.14
34    4 0 -1 52.93
35    4 1 -1 53.94
36    4 2 -1 54.95
37    4 3 -1 55.96
38    4 4 -1 56.97
39    4 5 -1 57.98
40    5 0 -1 65.50
41    5 1 -1 66.76
42    5 2 -1 68.03
43    5 3 -1 69.28
44    5 4 -1 70.54
45    5 5 -1 71.80
46    6 0 -1 69.19
47    6 1 -1 70.52
48    6 2 -1 71.86
49    6 3 -1 73.19
50    6 4 -1 74.52
51    6 5 -1 75.86
52    1 0 1 1 -1 28.01
53    1 0 1 2 -1 28.26
54    1 0 1 3 -1 28.51
55    1 0 1 4 -1 28.76
56    1 0 1 5 -1 29.02
57    1 1 1 2 -1 28.52
```

58	1	1	1	3	-1	28.76
59	1	1	1	4	-1	29.02
60	1	1	1	5	-1	29.27
61	1	2	1	3	-1	29.02
62	1	2	1	4	-1	29.27
63	1	2	1	5	-1	29.52
64	1	3	1	4	-1	29.52
65	1	3	1	5	-1	29.77
66	1	4	1	5	-1	30.03
67	2	0	1	1	-1	40.61
68	2	0	1	2	-1	40.86
69	2	0	1	3	-1	41.11
70	2	0	1	4	-1	41.36
71	2	0	1	5	-1	41.62
72	2	1	1	0	-1	40.86
73	2	1	1	2	-1	41.36
74	2	1	1	3	-1	41.62
75	2	1	1	4	-1	41.87
76	2	1	1	5	-1	42.12
77	2	2	1	0	-1	41.36
78	2	2	1	1	-1	41.62
79	2	2	1	3	-1	42.12
80	2	2	1	4	-1	42.37
81	2	2	1	5	-1	42.63
82	2	3	1	0	-1	41.87
83	2	3	1	1	-1	42.12
84	2	3	1	2	-1	42.37
85	2	3	1	4	-1	42.88
86	2	3	1	5	-1	43.13
87	2	4	1	0	-1	42.37
88	2	4	1	1	-1	42.63
89	2	4	1	2	-1	42.88
90	2	4	1	3	-1	43.13
91	2	4	1	5	-1	43.64
92	2	5	1	0	-1	42.88
93	2	5	1	1	-1	43.13
94	2	5	1	2	-1	43.38
95	2	5	1	3	-1	43.64
96	2	5	1	4	-1	43.89
97	5	0	1	1	-1	69.41
98	5	0	1	2	-1	69.63
99	5	0	1	3	-1	69.86
100	5	0	1	4	-1	70.08
101	5	0	1	5	-1	70.30
102	5	1	1	0	-1	70.30
103	5	1	1	2	-1	70.75
104	5	1	1	3	-1	70.97
105	5	1	1	4	-1	71.19
106	5	1	1	5	-1	71.41
107	5	2	1	0	-1	71.41
108	5	2	1	1	-1	71.63
109	5	2	1	3	-1	72.08
110	5	2	1	4	-1	72.33
111	5	2	1	5	-1	72.52
112	5	3	1	0	-1	72.52
113	5	3	1	1	-1	72.75
114	5	3	1	2	-1	72.97

115	5	3	1	4	-1	73.41		
116	5	3	1	5	-1	73.64		
117	5	4	1	0	-1	73.64		
118	5	4	1	1	-1	73.86		
119	5	4	1	2	-1	74.08		
120	5	4	1	3	-1	74.30		
121	5	4	1	5	-1	74.75		
122	5	5	1	0	-1	74.75		
123	5	5	1	1	-1	74.97		
124	5	5	1	2	-1	75.19		
125	5	5	1	3	-1	75.41		
126	5	5	1	4	-1	75.64		
127	4	0	2	1	-1	69.63		
128	4	0	2	2	-1	70.08		
129	4	0	2	3	-1	70.52		
130	4	0	2	4	-1	70.97		
131	4	0	2	5	-1	71.41		
132	4	1	2	0	-1	70.08		
133	4	1	2	2	-1	70.97		
134	4	1	2	3	-1	71.41		
135	4	1	2	4	-1	71.86		
136	4	1	2	5	-1	72.30		
137	4	2	2	0	-1	70.97		
138	4	2	2	1	-1	71.41		
139	4	2	2	3	-1	72.30		
140	4	2	2	4	-1	72.75		
141	4	2	2	5	-1	73.19		
142	4	3	2	0	-1	71.86		
143	4	3	2	1	-1	72.30		
144	4	3	2	2	-1	72.75		
145	4	3	2	4	-1	73.64		
146	4	3	2	5	-1	74.08		
147	4	4	2	0	-1	72.75		
148	4	4	2	1	-1	73.19		
149	4	4	2	2	-1	73.64		
150	4	4	2	3	-1	74.08		
151	4	4	2	5	-1	74.97		
152	4	5	2	0	-1	73.64		
153	4	5	2	1	-1	74.08		
154	4	5	2	2	-1	74.52		
155	4	5	2	3	-1	74.97		
156	4	5	2	4	-1	75.41		
157	1	0	1	1	1	2	-1	41.11
158	1	0	1	1	1	3	-1	41.36
159	1	0	1	1	1	4	-1	41.62
160	1	0	1	1	1	5	-1	41.87
161	1	0	1	2	1	3	-1	41.62
162	1	0	1	2	1	4	-1	41.87
163	1	0	1	2	1	5	-1	42.12
164	1	0	1	3	1	4	-1	42.12
165	1	0	1	3	1	5	-1	42.37
166	1	0	1	4	1	5	-1	42.62
167	1	1	1	2	1	3	-1	41.87
168	1	1	1	2	1	4	-1	42.12
169	1	1	1	2	1	5	-1	42.37
170	1	1	1	3	1	4	-1	42.37
171	1	1	1	3	1	5	-1	42.63

172 1 1 1 4 1 5 -1 42.88
173 1 2 1 3 1 4 -1 42.63
174 1 2 1 3 1 5 -1 42.88
175 1 2 1 4 1 5 -1 43.13
176 1 3 1 4 1 5 -1 43.38
177 1 0 1 1 1 2 1 3 -1 54.45
178 1 0 1 1 1 2 1 4 -1 54.7
179 1 0 1 1 1 2 1 5 -1 54.95
180 1 0 1 1 1 3 1 4 -1 54.95
181 1 0 1 1 1 3 1 5 -1 55.2
182 1 0 1 1 1 4 1 5 -1 55.46
183 1 0 1 2 1 3 1 4 -1 55.46
184 1 0 1 2 1 3 1 5 -1 55.46
185 1 0 1 2 1 4 1 5 -1 55.9
186 1 0 1 3 1 4 1 5 -1 56.1
187 1 0 1 2 1 3 1 4 -1 55.20
188 1 1 1 2 1 3 1 5 -1 55.71
189 1 1 1 2 1 3 1 4 -1 55.46
190 1 1 1 2 1 4 1 5 -1 55.9
191 1 1 1 3 1 4 1 5 -1 56.21
192 1 2 1 3 1 4 1 5 -1 56.46
193 1 0 1 1 1 2 1 3 1 4 -1 68.02
194 1 0 1 1 1 2 1 3 1 5 -1 68.27
195 1 0 1 1 1 2 1 4 1 5 -1 68.4
196 1 0 1 1 1 3 1 4 1 5 -1 69.0
197 1 0 1 2 1 3 1 4 1 5 -1 69.03
198 1 1 1 2 1 3 1 4 1 5 -1 69.28
199 1 0 1 1 1 2 1 3 1 4 1 5 -1 72.52
200 1 0 1 1 3 2 -1 67.26
201 1 0 1 1 4 2 -1 71.19
202 1 0 1 1 3 3 -1 68.02
203 1 0 1 1 3 4 -1 68.78
204 1 0 1 1 3 5 -1 69.53
205 1 0 1 1 4 3 -1 72.08
206 1 0 1 1 4 4 -1 72.97
207 1 0 1 1 4 5 -1 73.86
208 1 0 2 1 3 2 -1 70.97
209 1 0 2 1 3 3 -1 71.63
210 1 0 2 1 3 4 -1 72.3
211 1 0 3 1 1 2 -1 66.76
212 1 0 3 1 2 2 -1 70.75
213 1 0 3 1 1 3 -1 67.01
214 1 0 3 1 1 4 -1 67.26
215 1 0 3 1 1 5 -1 67.52
216 1 0 3 1 2 3 -1 71.19
217 1 0 3 1 2 4 -1 71.63
218 1 0 3 1 2 5 -1 72.08
219 1 0 4 1 1 2 -1 70.52
220 1 0 4 1 1 3 -1 70.75
221 1 0 4 1 1 4 -1 70.97
222 1 0 4 1 1 5 -1 71.19
223 1 0 1 2 3 3 -1 68.27
224 1 0 1 2 3 4 -1 68.94
225 1 0 1 2 3 5 -1 69.78
226 1 0 1 2 4 3 -1 72.3
227 1 0 1 2 4 4 -1 73.19
228 1 0 1 2 4 5 -1 74.08

229	1	0	2	2	3	3	-1	72.08
230	1	0	2	2	3	4	-1	72.75
231	1	0	3	2	1	3	-1	67.77
232	1	0	3	2	1	4	-1	68.02
233	1	0	3	2	1	5	-1	68.27
234	1	0	3	2	2	3	-1	71.86
235	1	0	3	2	2	4	-1	72.3
236	1	0	3	2	2	5	-1	72.75
237	1	0	4	2	1	3	-1	71.63
238	1	0	4	2	1	4	-1	71.86
239	1	0	4	2	1	5	-1	72.08
240	1	0	3	3	1	4	-1	68.78
241	1	0	3	3	1	5	-1	69.03
242	1	0	1	3	3	4	-1	69.28
243	1	0	3	4	1	5	-1	69.78
244	1	0	1	3	3	5	-1	70.04
245	1	0	1	4	3	5	-1	70.29
246	1	0	4	3	1	4	-1	72.75
247	1	0	3	3	2	4	-1	72.97
248	1	0	4	3	1	5	-1	72.97
249	1	0	2	3	3	4	-1	73.19
250	1	0	3	3	2	5	-1	73.41
251	1	0	1	3	4	4	-1	73.41
252	1	0	4	4	1	5	-1	73.86
253	1	0	3	4	2	5	-1	74.08
254	1	0	1	3	4	5	-1	74.3
255	1	0	1	4	4	5	-1	74.52
256	2	0	1	1	3	2	-1	70.75
257	2	0	1	1	3	3	-1	71.41
258	2	0	1	1	3	4	-1	72.08
259	2	0	3	1	1	2	-1	70.3
260	2	0	3	1	1	3	-1	70.52
261	2	0	3	1	1	4	-1	70.75
262	2	0	3	1	1	5	-1	70.97
263	2	0	1	2	3	3	-1	71.63
264	2	0	1	2	3	4	-1	72.3
265	2	0	3	2	1	3	-1	71.19
266	2	0	3	2	1	4	-1	71.41
267	2	0	3	2	1	5	-1	71.63
268	2	0	3	3	1	4	-1	72.08
269	2	0	3	3	1	5	-1	72.3
270	2	0	1	3	3	4	-1	72.52
271	2	0	3	4	1	5	-1	72.97
272	2	0	1	4	3	5	-1	73.4
273	3	0	1	1	1	2	-1	66.26
274	3	0	1	1	2	2	-1	70.3
275	3	0	1	1	1	3	-1	66.51
276	3	0	1	1	1	4	-1	66.76
277	3	0	1	1	1	5	-1	67.01
278	3	0	1	1	2	3	-1	70.75
279	3	0	1	1	2	4	-1	71.19
280	3	0	1	1	2	5	-1	71.63
281	3	0	2	1	1	2	-1	70.08
282	3	0	2	1	1	3	-1	70.3
283	3	0	2	1	1	4	-1	70.52
284	3	0	2	1	1	5	-1	70.75
285	3	0	1	2	1	3	-1	66.76

286	3	0	1	2	1	4	-1	67.01
287	3	0	1	2	1	5	-1	67.26
288	3	0	1	2	2	3	-1	70.97
289	3	0	1	2	2	4	-1	71.41
290	3	0	1	2	2	5	-1	71.86
291	3	0	2	2	1	3	-1	70.75
292	3	0	2	2	1	4	-1	70.97
293	3	0	2	2	1	5	-1	71.19
294	3	0	1	3	1	4	-1	67.26
295	3	0	1	3	1	5	-1	67.52
296	3	0	1	4	1	5	-1	69.77
297	3	0	2	3	1	4	-1	71.41
298	3	0	2	3	1	5	-1	71.63
299	3	0	1	3	2	4	-1	71.63
300	3	0	2	4	1	5	-1	72.08
301	3	0	1	4	2	5	-1	72.3
302	4	0	1	1	1	2	-1	69.26
303	4	0	1	1	1	3	-1	70.08
304	4	0	1	1	1	4	-1	70.3
305	4	0	1	1	1	5	-1	70.52
306	4	0	1	2	1	3	-1	70.3
307	4	0	1	2	1	4	-1	70.52
308	4	0	1	2	1	5	-1	70.75
309	4	0	1	3	1	4	-1	70.75
310	4	0	1	3	1	5	-1	70.97
311	4	0	1	4	1	5	-1	71.19
312	1	1	1	2	3	3	-1	68.52
313	1	1	1	2	3	4	-1	69.28
314	1	1	1	2	3	5	-1	70.04
315	1	1	1	2	4	3	-1	72.52
316	1	1	1	2	4	4	-1	73.41
317	1	1	1	2	4	5	-1	74.3
318	1	1	2	2	3	3	-1	72.3
319	1	1	2	2	3	4	-1	72.97
320	1	1	3	2	1	3	-1	68.02
321	1	1	3	2	1	4	-1	68.27
322	1	1	3	2	1	5	-1	68.52
323	1	1	3	2	2	3	-1	72.08
324	1	1	3	2	2	4	-1	72.52
325	1	1	3	2	2	5	-1	72.97
326	1	1	4	2	1	3	-1	71.86
327	1	1	4	2	1	4	-1	72.08
328	1	1	4	2	1	5	-1	72.3
329	1	1	3	3	1	4	-1	69.03
330	1	1	3	3	1	5	-1	69.28
331	1	1	1	3	3	4	-1	69.53
332	1	1	3	4	1	5	-1	70.04
333	1	1	1	3	3	5	-1	70.29
334	1	1	1	4	3	5	-1	70.54
335	1	1	4	3	1	4	-1	72.97
336	1	1	4	3	1	5	-1	73.19
337	1	1	3	3	2	4	-1	73.19
338	1	1	2	3	3	4	-1	73.41
339	1	1	1	3	4	4	-1	73.64
340	1	1	3	3	2	5	-1	73.64
341	1	1	4	4	1	5	-1	74.08
342	1	1	3	4	2	5	-1	74.3

343	1	1	1	3	4	5	-1	74.52
344	1	1	1	4	4	5	-1	74.75
345	2	1	1	2	3	3	-1	72.08
346	2	1	1	2	3	4	-1	72.75
347	2	1	3	2	1	3	-1	71.63
348	2	1	3	2	1	4	-1	71.86
349	2	1	3	2	1	5	-1	72.08
350	2	1	3	3	1	4	-1	72.52
351	2	1	3	3	1	5	-1	72.75
352	2	1	1	3	3	4	-1	72.97
353	2	1	3	4	1	5	-1	73.41
354	3	1	1	2	1	3	-1	67.52
355	3	1	1	2	1	4	-1	67.77
356	3	1	1	2	1	5	-1	68.02
357	3	1	1	2	2	3	-1	71.63
358	3	1	1	2	2	4	-1	72.08
359	3	1	1	2	2	5	-1	72.52
360	3	1	2	2	1	3	-1	71.41
361	3	1	2	2	1	4	-1	71.63
362	3	1	2	2	1	5	-1	71.86
363	3	1	1	3	1	4	-1	68.02
364	3	1	1	3	1	5	-1	68.27
365	3	1	1	4	1	5	-1	68.52
366	3	1	2	3	1	4	-1	72.08
367	3	1	2	3	1	5	-1	72.3
368	3	1	1	3	2	4	-1	72.3
369	3	1	2	4	1	5	-1	72.75
370	3	1	1	3	2	5	-1	72.75
371	3	1	1	4	2	5	-1	72.97
372	4	1	1	2	1	4	-1	71.41
373	4	1	1	2	1	5	-1	71.63
374	4	1	1	2	1	3	-1	71.19
375	4	1	1	3	1	4	-1	71.63
376	4	1	1	3	1	5	-1	71.86
377	4	1	1	4	1	5	-1	72.08
378	1	2	3	3	1	4	-1	69.28
379	1	2	3	3	1	5	-1	69.53
380	1	2	1	3	3	4	-1	69.78
381	1	2	3	4	1	5	-1	70.29
382	1	2	1	3	3	5	-1	70.54
383	1	2	1	4	3	5	-1	70.79
384	1	2	4	3	1	4	-1	73.19
385	1	2	4	3	1	5	-1	73.41
386	1	2	3	3	2	4	-1	73.41
387	1	2	2	3	3	4	-1	73.64
388	1	2	1	3	4	4	-1	73.86
389	1	2	3	3	2	5	-1	73.86
390	1	2	4	4	1	5	-1	74.3
391	1	2	3	4	2	5	-1	74.52
392	1	2	1	3	4	5	-1	74.75
393	1	2	1	4	4	5	-1	74.97
394	2	2	3	3	1	4	-1	72.97
395	2	2	3	3	1	5	-1	73.19
396	2	2	1	3	3	4	-1	73.41
397	2	2	3	4	1	5	-1	73.86
398	3	2	1	3	1	4	-1	68.78
399	3	2	1	3	1	5	-1	69.03

400	3	2	1	4	1	5	-1	69.28
401	3	2	2	3	1	4	-1	72.75
402	3	2	2	3	1	5	-1	72.97
403	3	2	1	3	2	4	-1	72.97
404	3	2	2	4	1	5	-1	73.41
405	3	2	1	3	2	5	-1	73.41
406	3	2	1	4	2	5	-1	73.64
407	4	2	1	3	1	4	-1	72.52
408	4	2	1	3	1	5	-1	72.75
409	4	2	1	4	1	5	-1	72.97
410	1	3	1	4	3	5	-1	71.04
411	1	3	1	4	4	5	-1	75.19
412	1	3	3	4	1	5	-1	70.54
413	1	3	3	4	2	5	-1	74.75
414	1	3	4	4	1	5	-1	74.52
415	2	3	3	4	1	5	-1	74.3
416	3	3	1	4	1	5	-1	69.04
417	3	3	1	4	2	5	-1	74.3
418	3	3	2	4	1	5	-1	74.08
419	4	3	1	4	1	5	-1	73.86
420	3	0	2	1	-1	65.77		
421	3	0	2	2	-1	66.01		
422	3	0	2	3	-1	66.77		
423	3	0	2	4	-1	67.26		
424	3	0	2	5	-1	67.77		
425	2	0	3	1	-1	66.01		
426	3	1	2	2	-1	67.26		
427	3	1	2	3	-1	67.77		
428	3	1	2	4	-1	68.27		
429	3	1	2	5	-1	68.78		
430	2	0	3	2	-1	67.01		
431	2	1	3	2	-1	67.52		
432	3	2	2	3	-1	68.52		
433	3	2	2	4	-1	69.03		
434	3	2	2	5	-1	69.53		
435	2	0	3	3	-1	67.77		
436	2	1	3	3	-1	68.27		
437	2	2	3	3	-1	68.78		
438	3	3	2	4	-1	69.78		
439	3	3	2	5	-1	70.29		
440	2	0	3	4	-1	68.52		
441	2	1	3	4	-1	69.03		
442	2	2	3	4	-1	69.53		
443	2	3	3	4	-1	70.03		
444	3	4	2	5	-1	71.04		
445	2	0	3	5	-1	69.23		
446	2	1	3	5	-1	69.84		
447	2	2	3	5	-1	70.35		
448	2	3	3	5	-1	70.86		
449	2	4	3	5	-1	71.37		
450	4	0	1	1	-1	65.76		
451	4	0	1	2	-1	66.01		
452	4	0	1	3	-1	66.26		
453	4	0	1	4	-1	66.51		
454	4	0	1	5	-1	66.76		
455	1	0	4	1	-1	66.51		
456	4	1	1	2	-1	67.01		

457 4 1 1 3 -1 67.26
458 4 1 1 4 -1 67.52
459 4 1 1 5 -1 67.77
460 1 0 4 2 -1 67.52
461 1 1 4 2 -1 67.77
462 4 2 1 3 -1 68.14
463 4 2 1 4 -1 68.52
464 4 2 1 5 -1 68.78
465 1 0 4 3 -1 68.52
466 1 1 4 3 -1 68.78
467 1 2 4 3 -1 69.14
468 4 3 1 4 -1 69.53
469 4 3 1 5 -1 69.78
470 1 0 4 4 -1 69.53
471 1 1 4 4 -1 69.78
472 1 2 4 4 -1 70.04
473 1 3 4 4 -1 70.30
474 4 4 1 5 -1 70.75
475 1 0 4 5 -1 70.52
476 1 1 4 5 -1 70.75
477 1 2 4 5 -1 70.97
478 1 3 4 5 -1 71.19
479 1 4 4 5 -1 71.41
480 2 0 2 1 1 2 -1 66.51
481 2 0 2 1 1 3 -1 66.76
482 2 0 2 1 1 4 -1 67.01
483 2 0 2 1 1 5 -1 67.26
484 1 0 2 1 2 2 -1 67.01
485 2 1 2 2 1 3 -1 67.77
486 2 1 2 2 1 4 -1 68.02
487 2 1 2 2 1 5 -1 68.27
488 1 0 2 2 2 3 -1 68.02
489 1 1 2 2 2 3 -1 68.27
490 2 2 2 3 1 4 -1 69.03
491 2 2 2 3 1 5 -1 69.28
492 1 0 2 3 2 4 -1 69.03
493 1 1 2 3 2 4 -1 69.28
494 1 2 2 3 2 4 -1 69.53
495 2 3 2 4 1 5 -1 70.29
496 1 0 2 4 2 5 -1 70.04
497 1 1 2 4 2 5 -1 70.29
498 1 2 2 4 2 5 -1 70.54
499 1 3 2 4 2 5 -1 70.79
500 2 0 1 1 2 2 -1 66.76
501 2 0 2 2 1 3 -1 67.26
502 2 0 2 2 1 4 -1 67.52
503 2 0 2 2 1 5 -1 67.77
504 2 0 1 1 2 3 -1 67.26
505 2 0 1 2 2 3 -1 67.51
506 2 0 2 3 1 4 -1 68.02
507 2 0 2 3 1 5 -1 68.27
508 2 0 1 1 2 4 -1 67.52
509 2 0 1 2 2 4 -1 68.02
510 2 0 1 3 2 4 -1 68.27
511 2 0 2 4 1 5 -1 68.78
512 2 0 1 1 2 5 -1 68.27
513 2 0 1 2 2 5 -1 68.52

514 2 0 1 3 2 5 -1 68.78
515 2 0 1 4 2 5 -1 69.03
516 1 0 2 1 2 3 -1 67.52
517 2 1 1 2 2 3 -1 68.02
518 2 1 2 3 1 4 -1 68.52
519 2 1 2 3 1 5 -1 68.78
520 1 0 2 1 2 4 -1 69.03
521 2 1 1 2 2 4 -1 68.52
522 2 1 1 3 2 4 -1 68.78
523 2 1 2 4 1 5 -1 69.28
524 1 0 2 1 2 5 -1 68.52
525 2 1 1 2 2 5 -1 69.03
526 2 1 1 3 2 5 -1 69.28
527 2 1 1 4 2 5 -1 69.53
528 1 0 2 2 2 4 -1 68.52
529 1 1 2 2 2 4 -1 68.78
530 2 2 1 3 2 4 -1 69.28
531 2 2 2 4 1 5 -1 69.78
532 1 0 2 2 2 5 -1 69.03
533 1 1 2 2 2 5 -1 69.28
534 2 2 1 3 2 5 -1 69.78
535 2 2 1 4 2 5 -1 70.04
536 1 0 2 3 2 5 -1 69.53
537 1 1 2 3 2 5 -1 69.78
538 1 2 2 3 2 5 -1 70.04
539 2 3 1 4 2 5 -1 70.30
540 2 0 2 1 -1 53.45
541 2 0 2 2 -1 53.93
542 2 0 2 3 -1 54.45
543 2 0 2 4 -1 54.95
544 2 0 2 5 -1 55.46
545 2 1 2 2 -1 54.45
546 2 1 2 3 -1 54.95
547 2 1 2 4 -1 55.46
548 2 1 2 5 -1 55.96
549 2 2 2 3 -1 55.46
550 2 2 2 4 -1 55.96
551 2 2 2 5 -1 56.46
552 2 3 2 4 -1 56.46
553 2 3 2 5 -1 56.97
554 2 4 2 5 -1 57.46
555 3 0 3 1 -1 69.86
556 3 0 3 2 -1 70.52
557 3 0 3 3 -1 71.19
558 3 0 3 4 -1 71.86
559 3 0 3 5 -1 72.52
560 3 1 3 2 -1 71.19
561 3 1 3 3 -1 71.86
562 3 1 3 4 -1 72.52
563 3 1 3 5 -1 73.19
564 3 2 3 3 -1 72.52
565 3 2 3 4 -1 73.19
566 3 2 3 5 -1 73.86
567 3 3 3 4 -1 73.86
568 3 3 3 5 -1 74.52
569 3 4 3 5 -1 75.19
570 3 0 1 1 -1 53.19

571 3 0 1 2 -1 53.44
572 3 0 1 3 -1 53.69
573 3 0 1 4 -1 53.94
574 3 0 1 5 -1 54.19
575 1 0 3 1 -1 53.69
576 3 1 1 2 -1 54.19
577 3 1 1 3 -1 54.45
578 3 1 1 4 -1 54.70
579 3 1 1 5 -1 54.95
580 1 0 3 2 -1 54.45
581 1 1 3 2 -1 54.70
582 3 2 1 3 -1 55.20
583 3 2 1 4 -1 55.46
584 3 2 1 5 -1 55.71
585 1 0 3 3 -1 55.20
586 1 1 3 3 -1 55.46
587 1 2 3 3 -1 55.71
588 3 3 1 4 -1 56.21
589 3 3 1 5 -1 56.46
590 1 0 3 4 -1 55.96
591 1 1 3 4 -1 56.21
592 1 2 3 4 -1 56.46
593 1 3 3 4 -1 56.72
594 3 4 1 5 -1 57.22
595 1 0 3 5 -1 56.72
596 1 1 3 5 -1 56.97
597 1 2 3 5 -1 57.22
598 1 3 3 5 -1 57.47
599 1 4 3 5 -1 57.72
600 2 0 2 1 1 2 1 3 -1 70.75
601 2 0 2 1 1 2 1 4 -1 70.97
602 2 0 2 1 1 2 1 5 -1 71.19
603 2 0 2 1 1 3 1 4 -1 71.19
604 2 0 2 1 1 3 1 5 -1 71.41
605 2 0 2 1 1 4 1 5 -1 71.63
606 2 0 1 1 2 2 1 3 -1 70.97
607 2 0 1 1 2 2 1 4 -1 71.19
608 2 0 1 1 2 2 1 5 -1 71.41
609 2 0 2 2 1 3 1 4 -1 71.63
610 2 0 2 2 1 3 1 5 -1 71.86
611 2 0 2 2 1 4 1 5 -1 72.08
612 2 0 1 1 1 2 2 3 -1 71.19
613 2 0 1 1 2 3 1 4 -1 71.63
614 2 0 1 1 2 3 1 5 -1 71.86
615 2 0 1 2 2 3 1 4 -1 71.86
616 2 0 1 2 2 3 1 5 -1 72.08
617 2 0 2 3 1 4 1 5 -1 72.52
618 2 0 1 1 1 2 2 4 -1 71.63
619 2 0 1 1 1 3 2 4 -1 71.86
620 2 0 1 1 2 4 1 5 -1 72.3
621 2 0 1 2 1 3 2 4 -1 72.08
622 2 0 1 2 2 4 1 5 -1 72.52
623 2 0 1 3 2 4 1 5 -1 72.75
624 2 0 1 1 1 2 2 5 -1 72.08
625 2 0 1 1 1 3 2 5 -1 72.3
626 2 0 1 1 1 4 2 5 -1 72.52
627 2 0 1 2 1 3 2 5 -1 72.52

628	2	0	1	2	1	4	2	5	-1	72.75
629	2	0	1	3	1	4	2	5	-1	72.97
630	1	0	2	1	2	2	1	3	-1	71.19
631	1	0	2	1	2	2	1	4	-1	71.41
632	1	0	2	1	2	2	1	5	-1	71.63
633	2	1	2	2	1	3	1	4	-1	72.08
634	2	1	2	2	1	3	1	5	-1	72.3
635	2	1	2	2	1	4	1	5	-1	72.52
636	1	0	2	1	1	2	2	3	-1	71.41
637	1	0	2	1	2	3	1	4	-1	71.86
638	1	0	2	1	2	3	1	5	-1	72.08
639	2	1	1	2	2	3	1	4	-1	72.3
640	2	1	1	2	2	3	1	5	-1	72.52
641	2	1	2	3	1	4	1	5	-1	72.97
642	1	0	2	1	1	2	2	4	-1	71.86
643	1	0	2	1	1	3	2	4	-1	72.08
644	1	0	2	1	2	4	1	5	-1	72.52
645	2	1	1	2	1	3	2	4	-1	72.52
646	2	1	1	2	2	4	1	5	-1	72.97
647	2	1	1	3	2	4	1	5	-1	73.19
648	1	0	2	1	1	2	2	5	-1	72.3
649	1	0	2	1	1	3	2	5	-1	72.52
650	1	0	2	1	1	4	2	5	-1	72.75
651	2	1	1	2	1	3	2	5	-1	72.97
652	2	1	1	2	1	4	2	5	-1	73.19
653	2	1	1	3	1	4	2	5	-1	73.41
654	1	0	1	1	2	2	2	3	-1	71.63
655	1	0	2	2	2	3	1	4	-1	72.3
656	1	0	2	2	2	3	1	5	-1	72.52
657	1	1	2	2	2	3	1	4	-1	72.52
658	1	1	2	2	2	3	1	5	-1	72.75
659	2	2	2	3	1	4	1	5	-1	73.41
660	1	0	1	1	2	2	2	4	-1	72.08
661	1	0	2	2	1	3	2	4	-1	72.52
662	1	0	2	2	2	4	1	5	-1	72.97
663	1	1	2	2	1	3	2	4	-1	72.75
664	1	1	2	2	2	4	1	5	-1	73.19
665	2	2	1	3	2	4	1	5	-1	73.64
666	1	0	1	1	2	2	2	5	-1	72.52
667	1	0	2	2	1	3	2	5	-1	72.97
668	1	0	2	2	1	4	2	5	-1	73.19
669	1	1	2	2	1	3	2	5	-1	73.19
670	1	1	2	2	1	4	2	5	-1	73.41
671	2	2	1	3	1	4	2	5	-1	73.86
672	1	0	1	1	2	3	2	4	-1	72.52
673	1	0	1	2	2	3	2	4	-1	72.75
674	1	0	2	3	2	4	1	5	-1	73.41
675	1	1	1	2	2	3	2	4	-1	72.97
676	1	1	2	3	2	4	1	5	-1	73.64
677	1	2	2	3	2	4	1	5	-1	73.86
678	1	0	1	1	2	3	2	5	-1	72.97
679	1	0	1	2	2	3	2	5	-1	73.19
680	1	0	2	3	1	4	2	5	-1	73.64
681	1	1	1	2	2	3	2	5	-1	73.41
682	1	1	2	3	1	4	2	5	-1	73.86
683	1	2	2	3	1	4	2	5	-1	74.08
684	1	0	1	1	2	4	2	5	-1	73.41

685	1	0	1	2	2	4	2	5	-1	73.64
686	1	0	1	3	2	4	2	5	-1	73.86
687	1	1	1	2	2	4	2	5	-1	73.86
688	1	1	1	3	2	4	2	5	-1	74.08
689	1	2	1	3	2	4	2	5	-1	74.3
690	2	0	1	1	1	2	1	3	-1	67.01
691	2	0	1	1	1	2	1	4	-1	67.26
692	2	0	1	1	1	2	1	5	-1	67.52
693	2	0	1	1	1	3	1	4	-1	67.52
694	2	0	1	1	1	3	1	5	-1	67.27
695	2	0	1	1	1	4	1	5	-1	68.02
696	2	0	1	2	1	3	1	4	-1	67.77
697	2	0	1	2	1	3	1	5	-1	68.02
698	2	0	1	2	1	4	1	5	-1	68.27
699	2	0	1	3	1	4	1	5	-1	68.52
700	1	0	2	1	1	2	1	3	-1	67.26
701	1	0	2	1	1	2	1	4	-1	67.52
702	1	0	2	1	1	2	1	5	-1	67.77
703	1	0	2	1	1	3	1	4	-1	67.77
704	1	0	2	1	1	3	1	5	-1	68.02
705	1	0	2	1	1	4	1	5	-1	68.27
706	2	1	1	2	1	3	1	4	-1	68.27
707	2	1	1	2	1	3	1	5	-1	68.52
708	2	1	1	2	1	4	1	5	-1	68.78
709	2	1	1	3	1	4	1	5	-1	69.03
710	1	0	1	1	2	2	1	3	-1	67.52
711	1	0	1	1	2	2	1	4	-1	67.77
712	1	0	1	1	2	2	1	5	-1	68.02
713	1	0	2	2	1	3	1	4	-1	68.27
714	1	0	2	2	1	3	1	5	-1	68.52
715	1	0	2	2	1	4	1	5	-1	68.78
716	1	1	2	2	1	3	1	4	-1	68.52
717	1	1	2	2	1	3	1	5	-1	68.78
718	1	1	2	2	1	4	1	5	-1	69.03
719	2	2	1	3	1	4	1	5	-1	69.53
720	1	0	1	1	1	2	2	3	-1	67.77
721	1	0	1	1	2	3	1	4	-1	68.27
722	1	0	1	1	2	3	1	5	-1	68.52
723	1	0	1	2	2	3	1	4	-1	68.52
724	1	0	1	2	2	3	1	5	-1	68.78
725	1	0	2	3	1	4	1	5	-1	69.28
726	1	1	1	2	2	3	1	4	-1	68.78
727	1	1	2	3	1	4	1	5	-1	69.53
728	1	2	2	3	1	4	1	5	-1	69.78
729	1	1	1	2	2	3	1	5	-1	69.03
730	1	0	1	1	1	2	2	4	-1	68.27
731	1	0	1	1	1	3	2	4	-1	68.52
732	1	0	1	1	2	4	1	5	-1	69.03
733	1	0	1	2	1	3	2	4	-1	68.78
734	1	0	1	2	2	4	1	5	-1	69.28
735	1	0	1	3	2	4	1	5	-1	69.53
736	1	1	1	2	1	3	2	4	-1	69.03
737	1	1	1	2	2	4	1	5	-1	69.53
738	1	1	1	3	2	4	1	5	-1	69.78
739	1	2	1	3	2	4	1	5	-1	70.04
740	1	0	1	1	1	2	2	5	-1	68.78
741	1	0	1	1	1	3	2	5	-1	69.03

742 1 0 1 1 1 4 2 5 -1 69.28
743 1 0 1 2 1 3 2 5 -1 69.28
744 1 0 1 2 1 4 2 5 -1 69.53
745 1 0 1 3 1 4 2 5 -1 69.78
746 1 1 1 2 1 3 2 5 -1 69.53
747 1 1 1 2 1 4 2 5 -1 69.78
748 1 1 1 3 1 4 2 5 -1 70.04
749 1 2 1 3 1 4 2 5 -1 70.29
750 2 0 1 1 1 2 1 3 1 4 -1 71.41
751 2 0 1 1 1 2 1 3 1 5 -1 71.63
752 2 0 1 1 1 2 1 4 1 5 -1 71.86
753 2 0 1 1 1 3 1 4 1 5 -1 72.08
754 2 0 1 2 1 3 1 4 1 5 -1 72.3
755 1 0 2 1 1 2 1 3 1 4 -1 71.63
756 1 0 2 1 1 2 1 3 1 5 -1 71.86
757 1 0 2 1 1 2 1 4 1 5 -1 72.08
758 1 0 2 1 1 3 1 4 1 5 -1 72.3
759 2 1 1 2 1 3 1 4 1 5 -1 72.75
760 1 0 1 1 2 2 1 3 1 4 -1 71.86
761 1 0 1 1 2 2 1 3 1 5 -1 72.08
762 1 0 1 1 2 2 1 4 1 5 -1 72.3
763 1 0 2 2 1 3 1 4 1 5 -1 72.75
764 1 1 2 2 1 3 1 4 1 5 -1 72.97
765 1 0 1 1 1 2 2 3 1 4 -1 72.08
766 1 0 1 1 1 2 2 3 1 5 -1 72.3
767 1 0 1 1 2 3 1 4 1 5 -1 72.75
768 1 0 1 2 2 3 1 4 1 5 -1 72.97
769 1 1 1 2 2 3 1 4 1 5 -1 73.19
770 1 0 1 1 1 2 1 3 2 4 -1 72.3
771 1 0 1 1 1 2 2 4 1 5 -1 72.75
772 1 0 1 1 1 3 2 4 1 5 -1 72.97
773 1 0 1 2 1 3 2 4 1 5 -1 73.19
774 1 1 1 2 1 3 2 4 1 5 -1 73.41
775 1 0 1 1 1 2 1 3 2 5 -1 72.75
776 1 0 1 1 1 2 1 4 2 5 -1 72.97
777 1 0 1 1 1 3 1 4 2 5 -1 73.19
778 1 0 1 2 1 3 1 4 2 5 -1 73.41
779 1 1 1 2 1 3 1 4 2 5 -1 73.64
780 3 0 1 1 1 2 1 3 -1 70.52
781 3 0 1 1 1 2 1 4 -1 70.75
782 3 0 1 1 1 2 1 5 -1 70.97
783 3 0 1 1 1 3 1 4 -1 70.97
784 3 0 1 1 1 3 1 5 -1 71.19
785 3 0 1 1 1 4 1 5 -1 71.41
786 3 0 1 2 1 3 1 4 -1 71.19
787 3 0 1 2 1 3 1 5 -1 71.41
788 3 0 1 2 1 4 1 5 -1 71.63
789 3 0 1 3 1 4 1 5 -1 71.86
790 1 0 3 1 1 2 1 3 -1 70.97
791 1 0 3 1 1 2 1 4 -1 71.19
792 1 0 3 1 1 2 1 5 -1 71.41
793 1 0 3 1 1 3 1 4 -1 71.41
794 1 0 3 1 1 3 1 5 -1 71.63
795 1 0 3 1 1 4 1 5 -1 71.86
796 3 1 1 2 1 3 1 4 -1 71.86
797 3 1 1 2 1 3 1 5 -1 72.08
798 3 1 1 2 1 4 1 5 -1 72.3

799	3	1	1	3	1	4	1	5	-1	72.52
800	1	0	1	1	3	2	1	3	-1	71.41
801	1	0	1	1	3	2	1	4	-1	71.63
802	1	0	1	1	3	2	1	5	-1	71.86
803	1	0	3	2	1	3	1	4	-1	72.08
804	1	0	3	2	1	3	1	5	-1	72.3
805	1	0	3	2	1	4	1	5	-1	72.52
806	1	1	3	2	1	3	1	4	-1	72.3
807	1	1	3	2	1	3	1	5	-1	72.52
808	1	1	3	2	1	4	1	5	-1	72.75
809	3	2	1	3	1	4	1	5	-1	73.19
810	1	0	1	1	1	2	3	3	-1	71.86
811	1	0	1	1	3	3	1	4	-1	72.3
812	1	0	1	1	3	3	1	5	-1	72.52
813	1	0	1	2	3	3	1	4	-1	72.52
814	1	0	1	2	3	3	1	5	-1	72.75
815	1	0	3	3	1	4	1	5	-1	73.19
816	1	1	1	2	3	3	1	4	-1	72.75
817	1	1	1	2	3	3	1	5	-1	72.97
818	1	2	3	3	1	4	1	5	-1	73.64
819	1	1	3	3	1	4	1	5	-1	73.41
820	1	0	1	1	1	2	3	4	-1	72.52
821	1	0	1	1	1	3	3	4	-1	72.75
822	1	0	1	1	3	4	1	5	-1	73.19
823	1	0	1	2	1	3	3	4	-1	72.97
824	1	0	1	2	3	4	1	5	-1	73.41
825	1	0	1	3	3	4	1	5	-1	73.64
826	1	1	1	2	1	3	3	4	-1	73.19
827	1	1	1	2	3	4	1	5	-1	73.64
828	1	1	1	3	3	4	1	5	-1	73.86
829	1	2	1	3	3	4	1	5	-1	74.08
830	1	0	1	1	1	2	3	5	-1	73.19
831	1	0	1	1	1	3	3	5	-1	73.41
832	1	0	1	1	1	4	3	5	-1	73.64
833	1	0	1	2	1	3	3	5	-1	73.64
834	1	0	1	2	1	4	3	5	-1	73.86
835	1	0	1	3	1	4	3	5	-1	74.08
836	1	1	1	2	1	3	3	5	-1	73.86
837	1	1	1	2	1	4	3	5	-1	74.08
838	1	2	1	3	1	4	3	5	-1	74.52
839	1	1	1	3	1	4	3	5	-1	74.3
840	2	0	1	1	1	2	-1	53.69		
841	2	0	1	1	1	3	-1	53.94		
842	2	0	1	1	1	4	-1	54.19		
843	2	0	1	1	1	5	-1	54.45		
844	2	0	1	2	1	3	-1	54.19		
845	2	0	1	2	1	4	-1	54.45		
846	2	0	1	2	1	5	-1	54.7		
847	2	0	1	3	1	4	-1	54.7		
848	2	0	1	3	1	5	-1	54.95		
849	2	0	1	4	1	5	-1	55.2		
850	1	0	2	1	1	2	-1	53.94		
851	1	0	2	1	1	3	-1	54.19		
852	1	0	2	1	1	4	-1	54.45		
853	1	0	2	1	1	5	-1	54.7		
854	2	1	1	2	1	3	-1	54.7		
855	2	1	1	2	1	4	-1	54.95		

856	2	1	1	2	1	5	-1	55.2
857	2	1	1	3	1	4	-1	55.2
858	2	1	1	3	1	5	-1	55.46
859	2	1	1	4	1	5	-1	55.71
860	1	0	1	1	2	2	-1	54.19
861	1	0	2	2	1	3	-1	54.7
862	1	0	2	2	1	4	-1	54.95
863	1	0	2	2	1	5	-1	55.2
864	1	1	2	2	1	3	-1	54.95
865	1	1	2	2	1	4	-1	55.2
866	1	1	2	2	1	5	-1	55.46
867	2	2	1	3	1	4	-1	55.71
868	2	2	1	3	1	5	-1	55.96
869	2	2	1	4	1	5	-1	56.21
870	1	0	1	1	2	3	-1	54.7
871	1	0	1	2	2	3	-1	54.95
872	1	0	2	3	1	4	-1	55.46
873	1	0	2	3	1	5	-1	55.71
874	1	1	1	2	2	3	-1	55.2
875	1	1	2	3	1	4	-1	55.71
876	1	1	2	3	1	5	-1	55.96
877	1	2	2	3	1	4	-1	55.96
878	1	2	2	3	1	5	-1	56.21
879	2	3	1	4	1	5	-1	56.72
880	1	0	1	1	2	4	-1	55.2
881	1	0	1	2	2	4	-1	55.46
882	1	0	1	3	2	4	-1	55.71
883	1	0	2	4	1	5	-1	56.21
884	1	1	1	2	2	4	-1	55.71
885	1	1	1	3	2	4	-1	55.96
886	1	1	2	4	1	5	-1	56.46
887	1	2	1	3	2	4	-1	56.21
888	1	2	2	4	1	5	-1	56.72
889	1	3	2	4	1	5	-1	56.97
890	1	0	1	1	2	5	-1	55.71
891	1	0	1	2	2	5	-1	55.96
892	1	0	1	3	2	5	-1	56.21
893	1	0	1	4	2	5	-1	56.46
894	1	1	1	2	2	5	-1	56.21
895	1	1	1	3	2	5	-1	56.46
896	1	1	1	4	2	5	-1	56.72
897	1	2	1	3	2	5	-1	56.72
898	1	2	1	4	2	5	-1	56.97
899	1	3	1	4	2	5	-1	57.22
900	1	2	2	3	3	5	-1	74.3
901	1	2	2	4	3	5	-1	74.75
902	2	0	1	3	3	5	-1	73.19
903	2	1	1	3	3	5	-1	73.64
904	2	2	1	3	3	5	-1	74.08
905	1	3	2	4	3	5	-1	74.97
906	2	0	1	4	3	5	-1	73.41
907	2	1	1	4	3	5	-1	73.41
908	2	2	1	4	3	5	-1	74.3
909	2	3	1	4	3	5	-1	74.75
910	1	0	2	1	3	5	-1	72.97
911	1	0	2	2	3	5	-1	73.41
912	1	0	2	3	3	5	-1	73.86

```
913    1 0 2 4 3 5 -1 74.3
914    2 0 1 1 3 5 -1 72.75
915    1 1 2 2 3 5 -1 73.64
916    1 1 2 3 3 5 -1 74.08
917    1 1 2 4 3 5 -1 74.52
918    2 0 1 2 3 5 -1 72.97
919    2 1 1 2 3 5 -1 73.41
920    2 0 2 1 2 2 -1 70.52
921    2 0 2 1 2 3 -1 70.97
922    2 0 2 1 2 4 -1 71.41
923    2 0 2 1 2 5 -1 71.86
924    2 0 2 2 2 3 -1 71.41
925    2 0 2 2 2 4 -1 71.86
926    2 0 2 2 2 5 -1 72.30
927    2 0 2 3 2 4 -1 72.30
928    2 0 2 3 2 5 -1 72.75
929    2 0 2 4 2 5 -1 73.19
930    2 1 2 2 2 3 -1 71.86
931    2 1 2 2 2 4 -1 72.30
932    2 1 2 2 2 5 -1 72.75
933    2 1 2 3 2 4 -1 72.75
934    2 1 2 3 2 5 -1 73.19
935    2 1 2 4 2 5 -1 73.64
936    2 2 2 3 2 4 -1 73.19
937    2 2 2 3 2 5 -1 73.64
938    2 2 2 4 2 5 -1 74.08
939    2 3 2 4 2 5 -1 74.52
940    -2
941
```

Appendix B Output File

The output file for the Savings and Cherry algorithms contains the inputs to the programs and the sections the algorithm created for the solution. The Improvement algorithm contains the inputs to the program, the beginning solution on which to improve, and the sections generated for the solution.

```
SAVINGS ALGORITHM

MAX PLY = 47  MAX # OF UNITS PER SECTION = 6
K = 1  INIT PLY = 1  Q = 1

ORDER
6 SIZE size-30
9 SIZE size-32
25 SIZE size-34
2 SIZE size-36
5 SIZE size-38
1 SIZE size-40


**********************************************
THE # OF FINAL SECTIONS ARE : 4
SECTION 0 HAS PLY = 1
                      AND 1 SIZE size-30
                      AND 4 SIZE size-38
                      AND 1 SIZE size-40
MARKER LENGTH =   73.86 THE TOTAL LENGTH =   73.86

SECTION 1 HAS PLY = 1
                      AND 1 SIZE size-32
                      AND 2 SIZE size-34
                      AND 2 SIZE size-36
                      AND 1 SIZE size-38
MARKER LENGTH =   72.52 THE TOTAL LENGTH =   72.52

SECTION 2 HAS PLY = 5
                      AND 1 SIZE size-30
                      AND 1 SIZE size-32
                      AND 4 SIZE size-34
MARKER LENGTH =   71.19 THE TOTAL LENGTH =  355.95

SECTION 3 HAS PLY = 3
                      AND 1 SIZE size-32
                      AND 1 SIZE size-34
MARKER LENGTH =   28.52 THE TOTAL LENGTH =   85.56

TOT MARKER =  246.09 TOT LENGTH =  587.89, UNIT OVER/UNDER = 0

TOTAL TIME = 0.008000 SECONDS
```

```
MAX PLY = 47  MAX # OF UNITS PER SECTION = 6
UNIT COST = 1 cents  CUT COST = 1 cents
ORDER
6 SIZE size-30
9 SIZE size-32
25 SIZE size-34
2 SIZE size-36
5 SIZE size-38
1 SIZE size-40

 FIRST SOLUTION
SECTION 0 HAS PLY = 6
       AND 1 SIZE size-30
       AND 1 SIZE size-32
       AND 4 SIZE size-34
MARKER LENGTH =   71.19  TOTAL LENGTH =  427.14

SECTION 1 HAS PLY = 2
       AND 1 SIZE size-32
       AND 1 SIZE size-36
       AND 2 SIZE size-38
MARKER LENGTH =   55.96  TOTAL LENGTH =  111.92

SECTION 2 HAS PLY = 1
       AND 1 SIZE size-32
       AND 1 SIZE size-34
       AND 1 SIZE size-38
       AND 1 SIZE size-40
MARKER LENGTH =   55.90  TOTAL LENGTH =   55.90

TOTAL MARKER =  183.05 TOTAL LENGTH =  594.96



**********************************************
THE # OF FINAL SECTIONS ARE : 7
SECTION 0 HAS PLY = 1
                    AND 1 SIZE size-30
                    AND 3 SIZE size-34
                    AND 1 SIZE size-38
                    AND 1 SIZE size-40
MARKER LENGTH =   72.52  TOTAL LENGTH =   72.52

SECTION 1 HAS PLY = 1
                    AND 3 SIZE size-32
                    AND 2 SIZE size-34
                    AND 1 SIZE size-36
MARKER LENGTH =   71.03  TOTAL LENGTH =   71.03

SECTION 2 HAS PLY = 1
                    AND 4 SIZE size-34
                    AND 1 SIZE size-36
                    AND 1 SIZE size-38
MARKER LENGTH =   72.52  TOTAL LENGTH =   72.52

SECTION 3 HAS PLY = 1
                    AND 1 SIZE size-32
                    AND 3 SIZE size-34
                    AND 2 SIZE size-38
MARKER LENGTH =   72.52  TOTAL LENGTH =   72.52

SECTION 4 HAS PLY = 1
                    AND 3 SIZE size-30
                    AND 2 SIZE size-32
                    AND 1 SIZE size-38
```

```
MARKER LENGTH =   70.52  TOTAL LENGTH =   70.52

SECTION 5 HAS PLY = 1
                    AND 1 SIZE size-32
                    AND 5 SIZE size-34
MARKER LENGTH =   71.63  TOTAL LENGTH =   71.63

SECTION 6 HAS PLY = 2
                    AND 1 SIZE size-30
                    AND 1 SIZE size-32
                    AND 4 SIZE size-34
MARKER LENGTH =   71.19  TOTAL LENGTH =  142.38

TOTAL MARKER =  501.93 TOTAL LENGTH =  573.12

UNIT OVER/UNDER = 0

TOTAL TIME = 0.020000
```

Appendix C Algorithm Detailed Descriptions

"Savings" Algorithm for COP

INPUT: (1) An order to be cut, consisting of the various sizes required and a quantity desired of each of these sizes. (2) The number of units over or under the demand that will be allowed. (3) The parameter k which determines the number of iterations after which the savings list will be updated. (4) The ply height of each of the initial sections. (5) List of l_i's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section). (6) Maximum ply height allowed. (7) Maximum number of sizes allowed per section. (8) The cutting cost per inch of fabric. (9) The unit cost of the fabric.

STEPS:

1. Assign each unit in the order to a separate section of the initial ply height.

2. Compute a *savings** achieved for combining any pair of sections into a single section. The maximum size of this list can be set to a specifc value. It is best to keep it less than or equal to the input K. The savings list is sorted as each value is calculated and placed in the list.

3. Start at the top of the savings list and *feasibly*** merge sections according the best savings. The first two sections that are merged are placed in a temporary section. Each merge thereafter is made only with this temporary section until the number of sizes per section is reached.

4. Once a the temporary section is full it is saved and cannot be used again.

5. After k mergers in step 3 the savings list should be updated and resorted by performing steps 2 and 3 for all newly created actions, then performing step 3. (note: k will be an input parameter)

6. Continue until no more savings can be achieved (i.e. the savings list has been scanned and the list is exhausted, with no mergers possible).

OUTPUT: (1) The number of sections, the sizes assigned to each of those sections, and the ply height of each section. (2) The total estimated fabric length required to cut the order. (3) The deviation of the number of units to be cut from the actual number of units required in the order.

*Savings Computations:

Step 2 of the algorithm requires a computation of savings achieved for combining two sections into one. Described below are the details of this computation, based on whether or not the two sections to be combined contain the same sizes or not.

Case A:

The two sections contain exactly the same size(s). The merger can be accomplished in one of two ways:

(i) Increase ply height by spreading one section on top of the other and making no change to the size combination in the section.

To compute the savings achieved in this situation, the cost savings is essentially based only on the cutting cost. That is, we need a number to reflect the savings of cutting the size combination in this section once instead of twice. (Note the length of fabric required for the section is the same before and after the merger and hence has no effect on the cost savings for the merger).

Let *e* represent the number of cutting inches in the pattern for the size combination in the two sections being considered. Then *e* is also the number of cutting inches required for the merged section as well. Recall that U = cutting cost/inch.

Thus, U*e* + U*e* = cost of cutting the two unmerged sections, and
U*e* = cost of cutting the merged sections. Hence,
U*e* = SAVINGS in cost obtained by merging the two sections. (Illustration attached).

However, the merger for case A could also be accomplished by

(ii) changing the size combination, leaving the ply height the same.

For example, suppose the two sections both contain sizes 32 and 34. The merged section will then contain the size combination 2-32s and 2-34s. Here the savings will be the decreased cost of fabric required for spreading the merged sections. Assume the following notation:

l_{i1} = length of fabric required to cut one layer of the 1st unmerged section,
l_{i2} = length of fabric required to cut one layer of the 2nd unmerged section, and
l_{i3} = length of fabric required to cut one layer of the 3rd MERGED section.
p = ply height of the unmerged and merged sections

Recall that c is the unit cost of fabric

Then, the savings can be computed as $cp(l_{i1} + l_{i2} - l_{i3})$. (Illustration attached).

Thus, for case A, the savings is the $\max\{Ue, cp(l_{i1} + l_{i2} - l_{i3})\}$.

If the ply heights of the two section are not equal and the second method of merging the two sections is better the following takes place:

Case B:

The two sections do not contain exactly the same size(s), but are of the same ply height. To maintain consistency, the only possible way to merge two such sections is to merge the size combination, leaving the ply height unchanged. This is precisely the same as case A(ii). Hence the savings computation is

$cp(l_{i1} + l_{i2} - l_{i3})$. (Illustration attached).

Case C:

The two sections do not contain the same size and have different ply heights. The only way to merge two such sections is to merge the size combination. This is the same as case B. Hence the savings computation is

$cp(l_{i1} + l_{i2} - l_{i3})$. (Illustration attached).

The ply height of the section being merged is chosen so that the minimum number of overages or underages are created.

**Feasibility Checks:

Step 4 of the algorithm states that section mergers should be done only when feasible. The feasibility of such mergers are based on two conditions:

(1) Will the maximum number of sizes allowed per section be violated? If so, do not merge.

(2) Will the maximum ply height be violated? If so, do not merge.

Cherry Picking Algorithm for COP

INPUT: (1) An order to be cut, consisting of the various sizes required and a specified demand quantity for each of these sizes. (2) The number of units over or under the demand that will be allowed. (3) Maximum ply height allowed. (4) Maximum number of sizes allowed per section. (5) List of l_i's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section).

STEPS:

1. Let q1 be the largest quantity of any size remaining in the order, and q2 be the second largest, where $q2 < q1$.

   (If there is no such q2, then one of two cases exists. Case 1: Only one size remains in the order, or Case 2: All sizes remaining have the same order quantity. In either case, set $q2 = q1$.)

   Form set S by selecting all sizes remaining in the order which have a quantity greater than or equal to q2 minus the number of units allowed over the specified demand.

2. The next section created will have ply height = min{q2, max ply height}. Combine the sizes in set S in this section in a way so that a minimal amount of fabric will be required, based on the inputs l_i. For example, if set S contains sizes small and large, it may be necessary to create two sections, one containing size small and the other size large, or only one section may be required which contains both sizes small and large. In the general case, all combinations of the sizes in set S should be considered which do not exceed the maximum number of sizes allowed per section.

3. Reduce the order demand quantities for the sizes in set S by q2.

4. If the order contains a size with positive quantity larger than the number of units allowed under the specified demand, go to step 1.

OUTPUT: (1) The number of sections, the sizes assigned to each of those sections, and the ply height of each section. (2) The total estimated fabric length required to cut the order. (3) The deviation of the number of units to be cut from the actual number of units required in the order.

Improvement Algorithm for COP

INPUT: (1) An order to be cut, consisting of the various sizes required and a quantity desired of each of these sizes. (2) The number of units over or under the demand that will be allowed. (3) A solution to the problem (see below for details) to be improved upon. (4) List of l_i's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section). (5) Maximum ply height allowed. (6) Maximum number of sizes allowed per section. (7) The cutting cost of the fabric per inch. (8) The unit cost of the fabric.

A SOLUTION consists of the following: (1) The number of sections, the sizes assigned to each of those sections, and the ply height of each section. (2) The total estimated fabric length required to cut the order. (3) The deviation of the number of units to be cut from the actual number of units required in the order.

OUTPUT: The output from the improvement algorithm will consist of a solution (as described above).

Step 0: We need to keep track of *starting over.* If we start over and cannot find any improvements after examining all possible exchanges, then the algorithm will terminate.

Each section contains one or more sizes. A portion of a section will consist of only one size. For example, if a section contains sizes M, M and L, the portions to consider are M, L, and MM.

STEPS:

Step 1. Consider the next portion of one section.

Step 2. Attempt to reassign the portion from its original section to one or more of the remaining sections so that the reassignment satisfies the feasibility checks listed below. If feasible to reassign, compute the savings that would be achieved by making the reassignment.

Step 3 Attempt to swap the portion from its original section with a portion from one of the remaining sections so that the reassignment satifies the feasibilty checks listed below. If feasible compute the savings that would be achieved by making the reassignment.

Step 4 Perform the reassignment based on the best savings computed.

How to perform the merger of the portion with a section and how to compute the associated savings can be described exactly as per the Savings Algorithm:

Case A:

The portion and section contain exactly the same size(s). The merger can be accomplished in one of two ways:

(i) Increase ply height by spreading one section on top of the other and making no change to the size combination in the section.

To compute the savings achieved in this situation, the cost savings is essentially based only on the cutting cost. That is, we need a number to reflect the savings of cutting the size combination in this section once instead of twice. (Note the length of fabric required for the section is the same before and after the merger and hence has no effect on the cost savings for the merger).

Let e_i represent the number of cutting inches in the pattern for the size combination in the two sections being considered. Then e_i is also the number of cutting inches required for the merged section as well. Recall that U = cutting cost/inch.

Thus, $Ue_i + Ue_i$ = cost of cutting the two unmerged sections, and
Ue_i = cost of cutting the merged sections. Hence,
Ue_i = SAVINGS in cost obtained by merging the two sections.

However, the merger for case A could also be accomplished by

(ii) changing the size combination, leaving the ply height the same.

For example, suppose the two sections both contain sizes 32 and 34. The merged section will then contain the size combination 2-32s and 2-34s. Here the savings will be the decreased cost of fabric required for spreading the merged sections. Assume the following notation:

l_{i1} = length of fabric required to cut one layer of the original section from which the portion will be cut (section A),
l_{i2} = length of fabric required to cut one layer of the candidate section into which the portion will be added (section B),
l_{i3} = length of fabric required to cut one layer of section A after the reassignment of the portion, and

l_{i4} = length of fabric required to cut one layer of section B after the reassignment of the portion.

p = ply height of the unmerged and merged sections
Recall that c is the unit cost of fabric

Then, the savings can be computed as $cp(l_{i1} + l_{i2} - l_{i3} - l_{i4})$.

Thus, for case A, the savings is the $\max\{Ue_i , cp(l_{i1} + l_{i2} - l_{i3} - l_{i4})\}$.

Case B:

The portion and section do not contain exactly the same size(s).

(i) Same ply height.
To maintain consistency, the only possible way to merge two such sections is to merge the size combination, leaving the ply height unchanged.

(ii) Ply heights not the same.
The merger should take place by combining the size combinations, and choosing the ply height so that the minimum number of overages or underages are created and all other feasbility checks are satisfied.

In either case (i) or (ii), we have the same situation as case A(ii). Hence the savings computation is

$$cp(l_{i1} + l_{i2} - l_{i3} - l_{i4}).$$

Feasibility Checks:

The feasibility of such mergers are based on two conditions:

(1) Will the maximum number of sizes allowed per section be violated? If so, do not merge.

(2) Will the maximum number of units over and under the demand be violated? If so, do not merge.

Appendix D Savings Algorithm Source Code

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/savings/case_ai.c    January 1991
3   -- ------------------------------------------------------------*/
4
5   /*------------------------------------------------------------
6   -  FILE NAME    : case_ai.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : January 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To compute the savings if ply heights and sizes
12  -       in both sections are the same.
13  -
14  -
15  -  MODIFICATION HISTORY-
16  -
17  - ------------------------------------------------------------*/
18  #include <stdio.h>
19  #include <stdlib.h>
20  #include "savedec.h"
21  #include "savelcl.h"
22
23  float case_ai(sect1, sect2, cut_cost)
24
25     section_t sect1;
26     section_t sect2;
27     int     cut_cost;
28
29  {
30     int i;
31     int e = 0;
32     float savings;
33
34     for (i=0; i< num_of_sizes; i++) {
35           e = e + (order.perimeter[i] * sect1.sizes[i]);
36           e = e + (order.perimeter[i] * sect2.sizes[i]);
37        }
38
39     savings = (float) cut_cost * e;
40
41     return(savings);
42
43  }
44
```

```
1   /* ------------------------------------------------------------------
2   -- $Header::   D:/cops/src/savings/case_aii.c    January 1991
3   -- ------------------------------------------------------------------*/
4
5   /*------------------------------------------------------------------
6   -  FILE NAME    : case_aii.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : January 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To compute the savings if the units or ply
12  -       height is not the same in two sections.
13  -
14  -
15  - ------------------------------------------------------------------*/
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include "savedec.h"
19  #include "savelcl.h"
20
21  float case_aii(sect1, sect2, unit_cost)
22
23     section_t sect1;
24     section_t sect2;
25     int  unit_cost;
26
27  {
28     int i;
29     int e = 0;
30     float savings;
31     float sect1_inch;
32     float sect2_inch;
33     float merge_inch;
34     order_t merged_order;
35
36     sect1_inch = find_inches(sect1.sizes);
37     sect2_inch = find_inches(sect2.sizes);
38
39     for (i=0; i< num_of_sizes; i++) {
40        merged_order[i] = 0;
41        merged_order[i] = merged_order[i] + sect1.sizes[i];
42        merged_order[i] = merged_order[i] + sect2.sizes[i];
43        }
44
45     merge_inch = find_inches(merged_order);
46
47     savings = unit_cost * sect1.ply_height * (sect1_inch + sect2_inch - merge_inch);
48
49     return(savings);
50
51  }
52
```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/savings/compute.c    January 1991
3   -- ------------------------------------------------------------*/
4
5   /*-------------------------------------------------------------
6   -  FILE NAME    : compute.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : January 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To determine which method to use to compute
12  -       the savings.
13  -
14  -
15  - ------------------------------------------------------------*/
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include <memory.h>
19  #include "savedec.h"
20  #include "savelcl.h"
21
22  float compute_savings(sect1, sect2, cut_cost, unit_cost, temp_save,
23                        max_sizes, max_ply)
24
25     section_t sect1;
26     section_t sect2;
27     int  cut_cost;
28     int  unit_cost;
29     savings_t *temp_save;
30     int max_sizes;
31     int max_ply;
32
33  {
34     int i;
35     int e = 0;
36     float savings = (float) 0.0;
37     float save2 = (float) 0.0;
38     char match = 1;
39     int num_units = 0;
40     int j, k, count;
41     char match2;
42
43     temp_save->ply1 = sect1.ply_height;
44     temp_save->ply2 = sect2.ply_height;
45
46     for (i=0; i<num_of_sizes; i++) {
47        if (sect1.sizes[i] != sect2.sizes[i])
48           match = 0;
49        num_units = num_units + sect1.sizes[i];
50        num_units = num_units + sect2.sizes[i];
51        }
52
53
54     if (match) {   /* sizes in sections are the same */
55        if ((sect1.ply_height + sect2.ply_height) <= max_ply) {
56           savings = case_ai(sect1, sect2, cut_cost);
57           temp_save->type= 1;
```

```
58            temp_save->ply_height = sect1.ply_height + sect2.ply_height;
59            }
60
61         else if (num_units <= max_sizes) {
62            save2 = case_aii(sect1, sect2, unit_cost);
63
64   /*          if ((save2 > savings) || (temp_save->ply_height > max_ply)) {*/
65               temp_save->type= 2;
66               savings = save2;
67
68               temp_save->ply_height = sect1.ply_height;
69              /* } */
70            }
71         }
72
73      else if ((sect1.ply_height == sect2.ply_height) && (num_units <= max_sizes)) {
74            savings = case_aii(sect1, sect2, unit_cost);
75            temp_save->type= 3;
76            temp_save->ply_height = sect1.ply_height;
77            }
78
79      else if (num_units <= max_sizes) {
80         temp_save->ply_height = sect1.ply_height;
81         savings = case_aii(sect1, sect2, unit_cost);
82         temp_save->type= 4;
83         }
84
85      temp_save->savings = savings;
86
87      return(savings);
88
89   }
90
```

```
1   /* ------------------------------------------------------------------
2   -- $Header::   D:/cops/src/savings/findinch.c     January 1991
3   -- ------------------------------------------------------------------*/
4
5   /*------------------------------------------------------------------
6   -  FILE NAME     : Findinch.c
7   -  PROGRAMMER    : Terri A. Smith
8   -  DATE WRITTEN : January 1991
9   -  ADDRESS       : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To find the length (in inches) in the list of Is
12  -
13  -
14  - ------------------------------------------------------------------*/
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include <string.h>
18  #include "savedec.h"
19  #include "savelcl.h"
20
21  float find_inches(sizes)
22
23     order_t sizes;
24
25  {
26     int i, j;
27     char match = 0;
28
29     i = 0;
30     while ((!match) && (i < num_list)) {
31        match = 1;
32        for (j=0; j<num_of_sizes; j++) {
33           if (sizes[j] != list[i].sizes[j])
34              match = 0;
35           }
36        ++i;
37        }
38
39     if (match)
40        return(list[--i].inches);
41     else {
42        printf(" COULDNT FIND ");
43        for (i=0; i<num_of_sizes; i++) {
44           if (sizes[i] > 0)
45              printf("%d %s ", sizes[i], order.ch_sizes[i]);
46           }
47        printf("\n");
48        exit(0);
49        }
50
51  }
52
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/savings/getparm.c    December 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Getparm.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : December 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- To read in the parameters from a file
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "savedec.h"
#include "savelcl.h"

int get_parameters(ou_units, max_ply, max_sizes,
                   k, init_ply, q, cut_cost, unit_cost)

   int   *ou_units;
   int   *max_ply;
   int   *max_sizes;
   int   *k;
   int   *init_ply;
   int   *q;
   int   *cut_cost;
   int   *unit_cost;

{
   int i, j, m, l;
   FILE *fp = NULL;
   int quantity;
   float temp;
   char match;

   if ((fp =fopen("INPUT", "r")) == NULL) {
      printf("Cannot open input file  - getparm.c");
      exit(0);
      }

   /* set order and list values to -1 */
   for (i = 0; i < MAX_SIZES; i++) {
      order.number[i] = 0;
      order.ch_sizes[i][0] = 0;
      order.perimeter[i] = 0;
      }

   for (i=0; i<MAX_LIST; i++) {
      list[i].inches = (float) 0.0;

      for (j = 0; j < MAX_SIZES; j++)
         list[i].sizes[j] = 0;
      }
```

```
58
59
60      fscanf(fp,"%d", ou_units);
61      fscanf(fp,"%d", max_ply);
62      fscanf(fp,"%d", max_sizes);
63      fscanf(fp,"%d", init_ply);
64      fscanf(fp,"%d", k);
65      fscanf(fp,"%d", cut_cost);
66      fscanf(fp,"%d", unit_cost);
67      fscanf(fp,"%d", q);
68
69
70      /* Input Order */
71      for (i = 0; i < MAX_SIZES; i++) {
72         fscanf(fp,"%d", &order.number[i]);
73         if (order.number[i] == -1) {
74            order.number[i] == 0;
75            break;
76            }
77
78         fscanf(fp,"%d", &order.perimeter[i]);
79         fscanf(fp,"%s", order.ch_sizes[i]);
80         }
81
82      num_of_sizes = i;
83
84
85      /* Input List */
86      i=0;
87      while(1) {
88
89        fscanf(fp,"%d", &quantity);
90
91        if (quantity == -2)
92            break;
93
94        while (quantity != -1) {
95
96            fscanf(fp,"%d", &m);
97
98            if (m >= num_of_sizes) {
99               printf("ERROR in reading size variable - getparm.c");
100              exit(0);
101              }
102
103           list[i].sizes[m] = quantity;
104
105           fscanf(fp,"%d", &quantity);
106           }
107
108       fscanf(fp,"%f", &list[i].inches);
109
110       ++i;
111       }
112
113     fclose(fp);
114
```

```
115         return(i);
116     }
117
```

```
1   /* ---------------------------------------------------------------
2   -- $Header::   D:/cops/src/savings/globals.h     January 1991
3   -- ---------------------------------------------------------------*/
4
5   /*---------------------------------------------------------------
6   -  FILE NAME     : Globals.h
7   -  PROGRAMMER    : Terri A. Smith
8   -  DATE WRITTEN : January 1991
9   -  ADDRESS       : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To declare all global variables
12  -
13  -
14  - ---------------------------------------------------------------*/
15  #include <stdio.h>
16  #include "savedec.h"
17  #include "savelcl.h"
18
19      ord_var_t order;
20
21      list_t    *list = NULL;
22
23      int      num_of_sizes;
24
25      int      num_list;
26
27      int      total_order = 0;
28
29      int      curr_tot = 0;
30
31      int      num_old_sect = 0;
32
33      section_t *old_sect = NULL;
```

```
1   INCLUDES = savedec.h
2   LIBNAME = savelib
3
4
5   OBJS = \
6           globals.obj \
7           getparm.obj \
8           findinch.obj \
9           case_ai.obj  \
10          case_aii.obj \
11          compute.obj
12
13
14  .c.obj:
15          $(CC)
16          $(LIB)
17
18
19  globals.obj : globals.c $(INCLUDES)
20
21  getparm.obj : getparm.c $(INCLUDES)
22
23  findinch.obj : findinch.c $(INCLUDES)
24
25  case_ai.obj : case_ai.c $(INCLUDES)
26
27  case_aii.obj : case_aii.c $(INCLUDES)
28
29  compute.obj : compute.c $(INCLUDES)
30
31  savings.obj : savings.c $(INCLUDES)
32
33  savings.exe : savings.obj  $(OBJS)
34          cl savings /link savelib.lib
35
36
37  $(B)\savings.exe : savings.exe
38          $(CP)
39
40  $(I)\savedec.h : savedec.h
41          $(CP)
42
43
```

```
/* ---------------------------------------------------------------
-- $Header::   D:/cops/src/savings/savedec.h    December 1990
-- ---------------------------------------------------------------*/

/*---------------------------------------------------------------
-  FILE NAME    : Savedec.h
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : December 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332   (404) 894-8952
-
-  PURPOSE- To define all variables and procedures
-
- ---------------------------------------------------------------*/
#ifndef SAVEDEC_H
#define SAVEDEC_H

#define MAX_LIST 1000
#define MAX_SIZES 25
#define MAX_SAVINGS 2


typedef int order_t[MAX_SIZES];

typedef char sizes_t[MAX_SIZES][10];

typedef struct {
   order_t   number;
   sizes_t   ch_sizes;
   int   perimeter[MAX_SIZES];
   } ord_var_t;

typedef struct {
   order_t  sizes;
   float    inches;
   } list_t;

typedef struct {
   order_t  sizes;
   int      ply_height;
   char     merged;
   } section_t;

typedef struct {
   int sect1;
   int sect2;
   int ply_height;
   float savings;
   int type;
   int ply1;
   int ply2;
   } savings_t;


int get_parameters(int *units, int *max_ply, int *max_sizes, int *k,
                   int *init_ply, int *q, int *cut_cost, int *unit_cost);

float find_inches(order_t sizes);
```

```
58
59    float case_ai(section_t sect1, section_t sect2, int cut_cost);
60
61    float case_aii(section_t sect1, section_t sect2, int unit_cost);
62
63    float compute_savings(section_t sect1, section_t sect2, int cut_cost,
64                int unit_cost, savings_t *temp_save, int max_sizes, int max_ply);
65
66
67    #endif
```

```
 /* ------------------------------------------------------------
-- $Header::   D:/cops/src/savings/savedec.h    December 1990
-- ------------------------------------------------------------*/

/*-------------------------------------------------------------
-  FILE NAME    : Savelcl.h
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : December 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332   (404) 894-8952
-
-  PURPOSE- To define all global variables
-
-
- -------------------------------------------------------------*/
#ifndef SAVELCL_H
#define SAVELCL_H


extern ord_var_t order;
extern list_t *list;
extern int num_list;
extern int num_of_sizes;
extern int total_order;
extern int curr_tot;
extern int num_old_sect;
extern section_t *old_sect;


#endif
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/savings/savings.c    December 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------------
-  FILE NAME    : Savings.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : December 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- Main program which controls execution of other procedures
-
-
- ------------------------------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include "savedec.h"
#include "savelcl.h"


#define clock() time(NULL)

main(argv, argc)
   int argv;
   char *argc[];

{
   /* Input Variables */
   int   ou_units;            /* # of units over/under allowed */
   int   max_ply;             /* max ply height allowed        */
   int   max_sizes;           /* # of sizes allowed / section  */
   int   init_ply;            /* initial ply height            */
   int   k;                   /* # of merges allowed           */
   int   cut_cost;            /* cutting cost / inch           */
   int   unit_cost;           /* unit cost                     */
   int   q;                   /* ply used for initial sections */

   /* Output Variables */
   float tot_length;          /* the total amt of fabric needed*/
   int   unit_dev;            /* deviation of units to cut from order */
   char  unit_string[10];     /* string for over, under        */
   float inches;              /* used in output                */

   int   i, j, x, y;            /* counters                     */
   int   curr_sect = 0;         /* current section              */
   section_t new_sect;          /* new sections                 */
   section_t *save_sect = NULL; /* new sections                 */
   div_t n;                   /* quotient and remainder        */
   savings_t temp_save;       /* temp savings - 1 structure    */
   savings_t *save_list = NULL; /* savings list                 */
   int   num_savings;         /* # of savings in list          */
   int   m, l, r, s;          /* counters                      */
```

```
58      char  mergers_possible = 1; /* Boolean for loop             */
59      int   num_new_sect;         /* # of total new_sect          */
60      int   num_save_sect;         /* # of total new_sect          */
61      int   num_units;            /* # of units in one section    */
62      int   unit_count;           /* # of units in all sections   */
63      int   order_count;          /* # of units in order          */
64      int   num_mergers = 0;      /* # of mergers                 */
65      FILE  *fp;                  /* output file pointer          */
66      clock_t start_time, end_time;  /* times                     */
67      double  total_time;         /* total time program runs      */
68      float  marker;              /* inches in marker             */
69      float  tot_marker;          /* total inches in all markers  */
70      char match2;                /* boolean value                */
71      int  count;                 /* counts the sections          */
72      int old_ply;                /* ply height of older section  */
73      section_t temp_sect;        /* temporary section            */
74      int add_sect;               /* # of sections to add         */
75      order_t temp_order;
76      order_t hold_sizes;
77      int abs1, abs2;
78
79      start_time = clock();
80
81      if ((fp = fopen("OUTPUT", "w")) == NULL) {
82         printf("CANNOT OPEN OUTPUT FILE    savings.c\n");
83         exit(0);
84         }
85
86      /*
87         Allocation of input list
88      */
89      if ((list = (list_t *)malloc(MAX_LIST * sizeof(list_t))) == NULL) {
90         printf("ALLOCATION ERROR FOR LIST    savings.c\n");
91         exit(0);
92         }
93
94      /*
95         Get parameters and print initial stuff to output file
96      */
97      num_list = get_parameters(&ou_units, &max_ply, &max_sizes, &k, &init_ply,
98                                &q, &cut_cost, &unit_cost);
99
100     fprintf(fp, "SAVINGS ALGORITHM 2\n\n");
101     fprintf(fp, "MAX PLY = %d  MAX # OF UNITS PER SECTION = %d\n", max_ply, max_sizes);
102    /* fprintf(fp, "UNIT COST = %d cents  CUT COST = %d cents\n", unit_cost, cut_cost);
103    */
104     fprintf(fp, "K = %d  INIT PLY = %d  Q = %d\n\n", k, init_ply, q);
105     fprintf(fp, "ORDER\n");
106     for (i=0; i<num_of_sizes; i++) {
107        fprintf(fp, "%d SIZE %s\n", order.number[i], order.ch_sizes[i]);
108        }
109
110     /*
111       Allocate space for two sets of sections
112     */
113     for (i=0; i<num_of_sizes; i++)
114        total_order = total_order + order.number[i];
```

```
115
116     if ((old_sect = (section_t *)malloc(total_order * sizeof(section_t))) == NULL) {
117        printf("ALLOCATION ERROR FOR OLD SECTION    savings.c\n");
118        exit(0);
119        }
120
121     if ((save_sect = (section_t *)malloc(total_order * sizeof(section_t))) == NULL) {
122        printf("ALLOCATION ERROR FOR SAVE SECTION    savings.c\n");
123        exit(0);
124        }
125
126     for (i=0; i<num_of_sizes; i++) {
127        temp_order[i] = order.number[i];
128        }
129
130     /*
131       Assign each unit in order to a separate section of initial
132       ply height
133     */
134     for (i =0; i < total_order; i++) {
135        old_sect[i].ply_height = q;
136        old_sect[i].merged = 0;
137
138        for (j=0; j< MAX_SIZES; j++)
139           old_sect[i].sizes[j] = 0;
140        }
141
142     unit_count = 0;
143     for (i=0; i<num_of_sizes; i++) {
144        n = div(order.number[i], q);
145        for (j=0; j<n.quot; j++) {
146           old_sect[curr_sect].sizes[i] = 1;
147           ++curr_sect;
148           unit_count = unit_count + q;
149           }
150        }
151
152     for (i=0; i<num_of_sizes; i++) {
153        n = div(order.number[i], q);
154        for (j=0; j<n.rem; j++) {
155           unit_dev = total_order - unit_count;
156           if ((ou_units - unit_dev) < 0) {
157              old_sect[curr_sect].sizes[i] = 1;
158              ++curr_sect;
159              unit_count = unit_count + q;
160              }
161           }
162        }
163
164     num_old_sect = curr_sect;
165
166     /*
167        Allocate space for savings list and initialize
168     */
169     if ((save_list = (savings_t *)malloc(MAX_SAVINGS * sizeof(savings_t))) == NULL) {
170        printf("ALLOCATION ERROR FOR SAVINGS LIST    savings.c\n");
171        exit(0);
```

```
172         }
173
174      for (i=0; i<MAX_SAVINGS; i++) {
175         save_list[i].savings = (float) 0.0;
176         save_list[i].ply_height = 0;
177         save_list[i].type = 0;
178         }
179
180
181      /*
182         Main loop in the Savings algorithm:
183            - creates a savings list and merges sections one at a time.
184            - a temporary section is created and merged with initial sections
185              until it is completely filled.  It is then save in the
186              save_section and a new temporary section is started
187            - When all sections are saved to the save_section then program is
188              terminated
189      */
190
191      num_save_sect = 0;
192      while (mergers_possible) {
193
194         for (i=0; i<MAX_SAVINGS; i++) {
195            save_list[i].savings = (float) 0.0;
196            save_list[i].ply_height = 0;
197            save_list[i].type = 0;
198            }
199
200
201         printf("NUM OLD SECT = %d\n", num_old_sect);
202         if (num_old_sect <= 1)
203            break;
204
205         num_units = 0;
206
207         /*
208            When the max number of units per section is reached, the section
209            is saved in save_section
210         */
211         for (j=0; j <num_of_sizes; j++)
212             num_units = old_sect[0].sizes[j] + num_units;
213
214         if (num_units >= max_sizes) {
215            memcpy(&save_sect[num_save_sect], &old_sect[0], sizeof(section_t));
216
217            for (i = 0; i<num_old_sect-1; i++)
218               memcpy(&old_sect[i], &old_sect[i+1], sizeof(section_t));
219
220            ++num_save_sect;
221            --num_old_sect;
222            }
223
224
225         mergers_possible = 0;
226         num_savings = 0;
227
228         /*
```

```
229           Create Savings List
230         */
231         i = 0;
232         for (j=i+1; j<num_old_sect; j++) {
233            temp_save.sect1 = i;
234            temp_save.sect2 = j;
235            temp_save.ply_height = 0;
236            temp_save.savings = (float) 0.0;
237            temp_save.type = 0;
238
239            compute_savings(old_sect[i], old_sect[j], cut_cost, unit_cost,
240                            &temp_save, max_sizes, max_ply);
241
242            m = 0;
243            while((m < num_savings) &&
244                  (temp_save.savings <= save_list[m].savings))
245               ++m;
246
247            if (m != MAX_SAVINGS) {
248
249               for (l = num_savings; l > m; l--) {
250                  memcpy(&save_list[l], &save_list[l-1], sizeof(savings_t));
251                  }
252
253               memcpy(&save_list[l], &temp_save, sizeof(savings_t));
254               if (num_savings < MAX_SAVINGS-1)
255                  ++num_savings;
256               }
257            }
258
259         /*
260           Merge Sections
261         */
262
263         new_sect.ply_height = q;
264         new_sect.merged = 0;
265
266         for (j=0; j< MAX_SIZES; j++)
267            new_sect.sizes[j] = 0;
268
269         num_mergers = 0;
270         m = 0;
271
272         for (i=0; i<num_savings; i++) {
273            r = save_list[i].sect1;
274            s = save_list[i].sect2;
275            num_units = 0;
276
277            for (j=0; j <num_of_sizes; j++) {
278               num_units = old_sect[r].sizes[j] + num_units;
279               if (save_list[i].type != 1)
280                  num_units = old_sect[s].sizes[j] + num_units;
281               }
282
283            if ((save_list[i].ply_height <= max_ply) &&
284                (!old_sect[r].merged) &&
285                (!old_sect[s].merged) &&
```

```
286                (num_units <= max_sizes) &&
287                (save_list[i].type != 0)) {
288
289                mergers_possible = 1;
290                old_sect[r].merged = 1;
291                old_sect[s].merged = 1;
292
293                new_sect.ply_height = save_list[i].ply_height;
294                for (j=0; j<num_of_sizes; j++) {
295                   new_sect.sizes[j] = new_sect.sizes[j] +
296                                       old_sect[r].sizes[j];
297                   if (save_list[i].type != 1)
298                        new_sect.sizes[j] = new_sect.sizes[j] +
299                                       old_sect[s].sizes[j];
300                   }
301
302                /*
303                   If the savings is achieved by rearranging sizes
304                   in one section (not by putting plys on top of
305                   each other), then the two ply heights of the sections
306                   must be manipulated to keep the order correct.
307                   e.g. If one section has ply 3 and the other ply 10
308                   one section of ply 3 with bothe sizes combinations
309                   is made and 7 sections of kept in the list of merging
310                   sections
311                */
312                if (save_list[i].type != 1) {
313
314                   for (x=0; x<num_of_sizes; x++)
315                      hold_sizes[x] = old_sect[s].sizes[x];
316
317                   /*
318                     Count how many sections in the sections list match
319                     the given section to merge
320                   */
321                   count = 0;
322                   for (l=1; l<num_old_sect; l++) {
323                      match2 = 1;
324                      for (j=0; j<num_of_sizes; j++) {
325                          if (old_sect[s].sizes[j] != old_sect[l].sizes[j])
326                             match2 = 0;
327                          }
328
329                      if (match2)
330                        ++count;
331                      }
332
333                   /*
334                      If the count is greater than the ply height of
335                      the temporary section, combine the two sections
336                      with the ply height of temporary section and then
337                      delete that number (ply height) of sections from
338                      the section list
339                   */
340                   if ((save_list[i].ply1 / q) <= count) {
341                      count = save_list[i].ply1 / q;
342                      for (l=1; l<num_old_sect; l++) {
```

```
343                     match2 = 1;
344                     for (j=0; j<num_of_sizes; j++) {
345                        if (hold_sizes[j] != old_sect[l].sizes[j])
346                           match2 = 0;
347                        }
348
349                     if ((match2) && (count > 0)) {
350                         for (m=l; m<num_old_sect-1; m++)
351                             memcpy(&old_sect[m], &old_sect[m+1], sizeof(section_t));
352
353                         --num_old_sect;
354                         --count;
355                         --l;
356                         }
357                     }
358                  } /* save_list[i].ply1 <= count */
359
360               /*
361                  else if the count is less than the ply height
362                  of the temporary section, then the temp section
363                  will have a ply height of count and sections are
364                  added back to the section list based on the the
365                  old_ply (of temp section) minus the the count
366               */
367               else {
368                  if (count > 0) {
369                     if (old_sect[0].ply_height > count) {
370                        old_ply = old_sect[0].ply_height;
371                        old_sect[0].ply_height = count;
372                        new_sect.ply_height = count;
373                        for (i=0; i<num_of_sizes; i++) {
374                           if (old_sect[0].sizes[i] > 0) {
375                               add_sect = old_ply - count;
376                               temp_sect.ply_height = q;
377                               temp_sect.merged = 0;
378
379                               for (j=0; j<num_of_sizes; j++)
380                                   temp_sect.sizes[j] = 0;
381
382                               temp_sect.sizes[i] = 1;
383
384                               for (l=0; l<old_sect[0].sizes[i]; l++) {
385                                   for (j=0; j<add_sect; j++)
386                                       memcpy(&old_sect[num_old_sect++], &temp_sect, sizeof(section_t));
387                                   }
388                               } /* if old_sect[0].sizes[i] > 0 */
389                           } /* for i=0 etc */
390                        } /* old_sect[0].ply > 0 */
391
392                     for (l=1; l<num_old_sect; l++) {
393                        match2 = 1;
394                        for (j=0; j<num_of_sizes; j++) {
395                           if (hold_sizes[j] != old_sect[l].sizes[j])
396                              match2 = 0;
397                           }
398
399                        if ((match2) && (count > 0)) {
```

```
400                             for (m=l; m<num_old_sect-1; m++)
401                                 memcpy(&old_sect[m], &old_sect[m+1], sizeof(section_t));
402
403                             --num_old_sect;
404                             --count;
405                             --l;
406                             }
407                           } /* for i=1 etc */
408
409                       } /* count > 0 */
410                     } /* else */
411                   }  /* if type != 1 */
412
413                 ++m;
414                 if (++num_mergers >= k)
415                    break;
416                 }
417               }
418
419             memcpy(&old_sect[0], &new_sect, sizeof(section_t));
420
421             /*
422                 Merges Complete
423             */
424
425             if (save_list[i].type == 1) {
426                for (i=s; i< num_old_sect-1; i++)
427                   memcpy(&old_sect[i], &old_sect[i+1], sizeof(section_t));
428
429                --num_old_sect;
430                }
431
432                num_savings = 0;
433
434
435          } /* End of While (1) */
436
437       if (num_old_sect > 0) {
438          for (i=0; i<num_old_sect; i++) {
439             memcpy(&save_sect[num_save_sect++], &old_sect[i], sizeof(section_t));
440             }
441          }
442
443       /*
444          put final information in output file
445       */
446       end_time = clock();
447       total_time = ((double) end_time - start_time) / CLK_TCK;
448
449       fprintf(fp, "\n\n*************************************************\n");
450       tot_length = (float) 0.0;
451       unit_dev = 0;
452       order_count = 0;
453       unit_count = 0;
454       tot_marker = (float) 0.0;
455
456       fprintf(fp, "THE # OF FINAL SECTIONS ARE : %d\n", num_save_sect);
```

```
457       for (i=0; i<num_save_sect; i++) {
458          fprintf(fp, "SECTION %d HAS PLY = %d\n", i, save_sect[i].ply_height);
459          for (j=0; j<num_of_sizes; j++) {
460             if (save_sect[i].sizes[j] > 0) {
461                fprintf(fp, "                    AND %d SIZE %s\n", save_sect[i].sizes[j], order.ch_sizes[j]);
462            unit_count = unit_count + (save_sect[i].sizes[j] * save_sect[i].ply_height);
463                }
464             }
465          marker = find_inches(save_sect[i].sizes);
466          inches = marker * save_sect[i].ply_height;
467          fprintf(fp, "MARKER LENGTH = %7.2f THE TOTAL LENGTH = %7.2f\n\n",
468                  marker, inches);
469          tot_length = tot_length + inches;
470          tot_marker = tot_marker + marker;
471          }
472
473       for (j=0; j<num_of_sizes; j++)
474            order_count = order_count + order.number[j];
475
476       unit_dev = order_count - unit_count;
477       if (unit_dev > 0)
478          strcpy(unit_string, "UNDER");
479       else if (unit_dev == 0)
480          strcpy(unit_string, "\0");
481       else {
482          unit_dev = unit_dev * -1;
483          strcpy(unit_string, "OVER");
484          }
485
486       fprintf(fp, "TOT MARKER = %7.2f TOT LENGTH = %7.2f, UNIT OVER/UNDER = %d %s\n\n",
487               tot_marker, tot_length, unit_dev, unit_string);
488       fprintf(fp, "TOTAL TIME = %f SECONDS\n", total_time);
489
490
491       /*
492          Free all space and close output file
493       */
494       if (list != NULL)
495          free(list);
496
497       if (save_list != NULL)
498          free(save_list);
499
500       if (old_sect != NULL)
501          free(old_sect);
502
503
504       fclose(fp);
505
506       return(0);
507    }
```

Appendix E Cherry Algorithm Source Code

```
1   /* ---------------------------------------------------------------
2   -- $Header::   D:/cops/src/cherry/cherdec.h    December 1990
3   -- ---------------------------------------------------------------*/
4
5   /*---------------------------------------------------------------
6   -  FILE NAME    : Cherdec.h
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : December 1990
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To define all variables and procedures
12  -
13  -
14  - ---------------------------------------------------------------*/
15  #ifndef CHERDEC_H
16  #define CHERDEC_H
17
18  #define MAX_LIST 1000
19  #define MAX_SIZES 25
20
21  typedef int order_t[MAX_SIZES];
22
23  typedef char sizes_t[MAX_SIZES][10];
24
25  typedef struct {
26     order_t  number;
27     sizes_t  ch_sizes;
28     } ord_var_t;
29
30  typedef struct {
31     order_t sizes;
32     float inches;
33     } list_t;
34
35  typedef struct {
36     order_t sizes;
37     int     ply_height;
38     } section_t;
39
40  int get_parameters(int *units, int *max_ply, int *max_sizes);
41
42  float find_inches(order_t sizes);
43
44  float combine_inches(order_t set_s);
45
46  void check_inches(section_t *temp_secs, int *num_temp_secs);
47
48  void clear_temp(section_t *temp_secs, int *num_temp_secs);
49
50  void copy_hold_to_sections();
51
52  void ones(order_t set_s, section_t *temp_secs, int *num_temp_secs);
53
54  void twos(order_t set_s, section_t *temp_secs, int *num_temp_secs);
55
56  void threes(order_t set_s, section_t *temp_secs, int *num_temp_secs);
57
```

```
58    void fours(order_t set_s, section_t *temp_secs, int *num_temp_secs);
59
60    void fives(order_t set_s, section_t *temp_secs, int *num_temp_secs);
61
62    void sixes(order_t set_s, section_t *temp_secs, int *num_temp_secs);
63
64    #endif
```

```
 /* ------------------------------------------------------------
-- $Header::   D:/cops/src/savings/savedec.h    December 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------------
-  FILE NAME    : Savelcl.h
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : December 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- To define all global variables
-
-
- ------------------------------------------------------------------*/
#ifndef CHERLCL_H
#define CHERLCL_H


extern ord_var_t order;
extern list_t *list;
extern int num_list;
extern int num_of_sizes;
extern order_t temp_order;
extern int num_sections;
extern float total_inches;
extern float prev_inch;
extern section_t *sections;
extern int num_hold_secs;
extern section_t *hold_secs;
extern int ply_height;


#endif
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/cherry.c    December 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Cherry.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : December 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- The main program which executes all other procedures
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <time.h>
#include "cherdec.h"
#include "cherlcl.h"

#define clock() time(NULL)

main(argv, argc)
   int argv;
   char *argc[];

{
   /* Input Variables */
   int     ou_units;        /* # of units over/under allowed */
   int     max_ply;         /* max ply height                */
   int     max_sizes;       /* # of sizes allowed / section  */

   /* Output Variables */
   int     num_temp_secs;   /* # of total sections           */
   float  tot_length;      /* total length of fabric        */
   float  tot_marker;      /* total length of fabric        */
   section_t *temp_secs=NULL; /* each section description    */
   int     unit_dev = 0;    /* deviation of # of units from order */
   char    unit_string[10]; /* string to print over, under  */

   int     q1;              /* largest quantity in order     */
   int     q2;              /* 2nd largest quantity in order */
   int     s_var;           /* q2 minus ou_units             */
   order_t set_s;           /* set S of sizes                */
   int     max_sections=0;  /* max # of sections to allocate */
   char    repeat_loop;     /* boolean to loop again or not  */
   int     i, j, k, l,m,n;  /* counters                      */
   float  inches;          /* used for printing results     */
   float  marker;          /* used for printing results     */
   int     sets_cnt;        /* # of sizes in set S           */
   FILE    *fp;             /* output file pointer           */
   int     unit_count = 0;  /* # of units in all sections    */
   int     order_count = 0; /* # of units in order           */
   int     ou_count = 0;    /* count to determine if repeat loop */
```

```
58      clock_t start_time, end_time;
59      double total_time;
60
61      start_time = clock();
62
63      /*
64        Open output file
65      */
66      if ((fp = fopen ("OUTPUT", "w")) == NULL ) {
67         printf("CANNOT OPEN OUTPUT FILE   cherry.c\n");
68         exit(0);
69         }
70
71      /*
72         Allocate space for the list of Is
73      */
74      if ((list = (list_t *)malloc(MAX_LIST * sizeof(list_t))) == NULL) {
75         printf("ALLOCATION ERROR FOR LIST    cherry.c\n");
76         exit(0);
77         }
78
79      num_list = get_parameters(&ou_units, &max_ply, &max_sizes);
80
81      fprintf(fp, "CHERRY ALGORITHM\n\n");
82      fprintf(fp, "MAX PLY = %d  MAX # OF UNITS PER SECTION = %d\n", max_ply, max_sizes);
83      fprintf(fp, "\n ORDER\n");
84      for (i=0; i<num_of_sizes; i++) {
85         fprintf(fp, "%d SIZE %s\n", order.number[i], order.ch_sizes[i]);
86         order_count = order_count + order.number[i];
87         }
88
89      /*
90         Allocate space for the max number of sections
91         for the three list of sections
92      */
93      for (i=0; i< MAX_SIZES; i++) {
94         max_sections = max_sections + order.number[i];
95         }
96
97      if ((sections = (section_t *)malloc(max_sections * sizeof(section_t))) == NULL) {
98         printf("ALLOCATION ERROR FOR SECTIONS    cherry.c\n");
99         exit(0);
100        }
101
102     if ((temp_secs = (section_t *)malloc(max_sections * sizeof(section_t))) == NULL) {
103        printf("ALLOCATION ERROR FOR SECTIONS    cherry.c\n");
104        exit(0);
105        }
106
107     if ((hold_secs = (section_t *)malloc(max_sections * sizeof(section_t))) == NULL) {
108        printf("ALLOCATION ERROR FOR SECTIONS    cherry.c\n");
109        exit(0);
110        }
111
112
113     for (i=0; i<max_sections; i++) {
114        sections[i].ply_height = 0;
```

```
115         for (j=0; j<MAX_SIZES; j++)
116            sections[i].sizes[j] = 0;
117         }
118
119      num_sections = 0;
120
121      /*
122         Main Loop of program
123      */
124      while (1) {
125
126         for (i=0; i<max_sections; i++) {
127            temp_secs[i].ply_height = 0;
128            for (j=0; j<MAX_SIZES; j++)
129               temp_secs[i].sizes[j] = 0;
130            }
131
132         repeat_loop = 0;
133
134         /*
135            Choose Q1 and Q2
136         */
137         q1 = 0;
138         q2 = 0;
139
140         for (i=1; i<num_of_sizes; i++) {
141            if (order.number[i] > order.number[q1])
142               q1 = i;
143            }
144
145         q2 =0;
146         for (i=0; i<num_of_sizes; i++) {
147            if (i!= q1) {
148               if (order.number[i] >= 0) {
149                  q2 = i;
150                  break;
151                  }
152               }
153            }
154
155         for (i=0; i<num_of_sizes; i++) {
156            if (i != q1)
157               if (order.number[i] >= order.number[q2])
158                  q2 = i;
159            }
160
161         if (order.number[q2] <= 0)
162            q2 = q1;
163
164
165         /*
166           Form set S with all the sizes remaining in the order
167           which have a quantity greater than or equal to q2 - the number
168           of units allowed over the specified demand
169         */
170         s_var = order.number[q2] - ou_units;
171
```

```
172         sets_cnt = 0;
173         for (i=0; i<MAX_SIZES; i++) {
174            if ((order.number[i] >= s_var) && (order.number > 0)) {
175                set_s[i] = 1;
176                ++sets_cnt;
177                }
178            else
179                set_s[i] = 0;
180             }
181
182          /*
183            Set ply height of next section to the min(q2, max ply)
184          */
185          ply_height = order.number[q2];
186          if (max_ply < order.number[q2])
187             ply_height = max_ply;
188
189
190          /*
191             Combine all posibilities of sections up to 5 units
192             per section
193          */
194          inches = (float) 9999.0;
195          for (i=0; i<MAX_SIZES; i++)
196                temp_order[i] = 0;
197          num_temp_secs = 0;
198
199          total_inches = (float) 0.0;
200
201
202          ones(set_s, temp_secs, &num_temp_secs);
203          check_inches(temp_secs, &num_temp_secs);
204          clear_temp(temp_secs, &num_temp_secs);
205
206          if ((sets_cnt > 1) && (max_sizes > 1)) {
207             twos(set_s, temp_secs, &num_temp_secs);
208             check_inches(temp_secs, &num_temp_secs);
209             clear_temp(temp_secs, &num_temp_secs);
210             }
211
212          if ((sets_cnt > 2) && (max_sizes > 2)) {
213             threes(set_s, temp_secs, &num_temp_secs);
214             check_inches(temp_secs, &num_temp_secs);
215             clear_temp(temp_secs, &num_temp_secs);
216             }
217
218          if ((sets_cnt > 3) && (max_sizes > 3)) {
219             fours(set_s, temp_secs, &num_temp_secs);
220             check_inches(temp_secs, &num_temp_secs);
221             clear_temp(temp_secs, &num_temp_secs);
222             }
223
224          if ((sets_cnt > 4) && (max_sizes > 4)) {
225             fives(set_s, temp_secs, &num_temp_secs);
226             check_inches(temp_secs, &num_temp_secs);
227             clear_temp(temp_secs, &num_temp_secs);
228             }
```

```
229
230         if ((sets_cnt > 5) && (max_sizes > 5)) {
231            sixes(set_s, temp_secs, &num_temp_secs);
232            check_inches(temp_secs, &num_temp_secs);
233            clear_temp(temp_secs, &num_temp_secs);
234            }
235
236         copy_hold_to_sections();
237
238         /*
239           Reduce the order demand
240         */
241         for (m=(num_sections - num_hold_secs); m<num_sections; m++) {
242            for (n=0; n< num_of_sizes; n++) {
243               if (sections[m].sizes[n] == 1) {
244                  order.number[n] = order.number[n] - ply_height;
245                  set_s[n] = 0;
246                  }
247               }
248            }
249
250         /*
251           Repeat loop if the order contains a size w/ positive
252           quantity greater than the number of units allowed under the
253           specified demand, else break out of loop
254         */
255
256         ou_count = 0;
257         for (i=0; i<num_of_sizes; i++) {
258            ou_count = ou_count + order.number[i];
259            if (ou_count > ou_units)
260               repeat_loop = 1;
261            }
262
263         if (!repeat_loop)
264            break;
265
266         } /* END of While (1) */
267
268      end_time = clock();
269      total_time = ((double) end_time - start_time) / CLK_TCK;
270
271      /*
272         Print Out Results
273      */
274      fprintf(fp, "\n\n*********************************************\n");
275      fprintf(fp, "THE NUMBER OF FINAL SECTIONS = %d\n", num_sections);
276
277      for (i=0; i<num_sections; i++) {
278         marker = find_inches(sections[i].sizes);
279         inches = marker * sections[i].ply_height;
280         total_inches = total_inches + inches;
281         tot_marker = tot_marker + marker;
282         fprintf(fp, "\nSECTION %d HAS PLY = %d\n", i, sections[i].ply_height);
283         for (j=0; j<num_of_sizes; j++) {
284            if (sections[i].sizes[j] > 0) {
285               fprintf(fp, "          HAS %d SIZE %s\n", sections[i].sizes[j], order.ch_sizes[j]);
```

```
286                unit_count = unit_count + (sections[i].sizes[j] * sections[i].ply_height);
287                }
288             }
289          fprintf(fp, "MARKER INCHES = %7.2f and TOTAL INCHES %7.2f\n", marker, inches);
290          }
291       fprintf(fp, "\nTOTAL MARKER INCHES = %7.2f  TOTAL INCHES = %7.2f\n", tot_marker, total_inches);
292
293
294       unit_dev = order_count - unit_count;
295       if (unit_dev > 0)
296          strcpy(unit_string, "UNDER");
297       else if (unit_dev == 0)
298          strcpy(unit_string, "\0");
299       else {
300          unit_dev = unit_dev * -1;
301          strcpy(unit_string, "OVER");
302          }
303
304       fprintf(fp, "UNIT OVER/UNDER = %d %s\n\n", unit_dev, unit_string);
305       fprintf(fp, "TOTAL_TIME = %f\n", total_time);
306
307
308       if (list != NULL)
309          free(list);
310
311       if (sections != NULL)
312          free(sections);
313
314       if (temp_secs != NULL)
315          free(temp_secs);
316
317       if (hold_secs != NULL)
318          free(hold_secs);
319
320       fclose(fp);
321
322       return(0);
323    }
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/chkinch.c    December 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Chkinch.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : December 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- To determine if the total inches calculated from
-       the last grouping of sections is less than any previous
-       grouping.  If so the sections are saved in the hold
-       sections.
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <memory.h>
#include "cherdec.h"
#include "cherlcl.h"

void check_inches(temp_secs, num_temp_secs)
   section_t *temp_secs;
   int *num_temp_secs;

{

   int m, i, j;

   if ((total_inches < prev_inch) && (total_inches > (float) 0.0)) {
      num_hold_secs = 0;
      for (m=0; m<*num_temp_secs; m++) {
         memcpy(&hold_secs[num_hold_secs], &temp_secs[m], sizeof(section_t));
         hold_secs[num_hold_secs].ply_height = ply_height;
         ++num_hold_secs;
         }
      prev_inch = total_inches;
      }

}
```

```
1   /* ------------------------------------------------------------------
2   -- $Header::   D:/cops/src/cherry/clrtemp.c    December 1990
3   -- ------------------------------------------------------------------*/
4
5   /*------------------------------------------------------------------
6   -  FILE NAME    : Clrtemp.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : January 1990
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- Initializes the temp sections
12  -
13  -
14  - ------------------------------------------------------------------*/
15  #include <stdio.h>
16  #include <malloc.h>
17  #include <stdlib.h>
18  #include <memory.h>
19  #include "cherdec.h"
20  #include "cherlcl.h"
21
22  void clear_temp(temp_secs, num_temp_secs)
23     section_t *temp_secs;
24     int *num_temp_secs;
25
26  {
27     int i, j;
28
29     total_inches = (float) 0.0;
30     for (i=0; i< *num_temp_secs; i++) {
31        for (j=0; j< num_of_sizes; j++) {
32           temp_secs[i].sizes[j] = 0;
33           }
34        }
35     *num_temp_secs = 0;
36
37  }
```

```
/* ---------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/combine.c    January 1991
-- ---------------------------------------------------------------*/

/*---------------------------------------------------------------
-  FILE NAME    : Combine.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : January 1991
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- Finds the length (in inches) of the combine units
-       in one section.
-
-
- ---------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherlcl.h"

float combine_inches(temp_order)

   order_t temp_order;

{
   float inches;

   inches = find_inches(temp_order);

   return(inches);
}

```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/cherry/cphold.c    December 1990
3   -- ------------------------------------------------------------*/
4
5   /*-------------------------------------------------------------
6   -  FILE NAME    : Cphold.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : December 1990
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- Copies the temp sections into the hold sections.
12  -
13  - -------------------------------------------------------------*/
14  #include <stdio.h>
15  #include <malloc.h>
16  #include <stdlib.h>
17  #include <memory.h>
18  #include "cherdec.h"
19  #include "cherlcl.h"
20
21  void copy_hold_to_sections()
22
23  {
24
25     int m;
26
27     for (m=0; m<num_hold_secs; m++) {
28        memcpy(&sections[num_sections], &hold_secs[m], sizeof(section_t));
29        sections[num_sections].ply_height = ply_height;
30        ++num_sections;
31        }
32
33     prev_inch = (float) 9999.0;
34  }
```

```
/* ---------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/findinch.c     January 1991
-- ---------------------------------------------------------------*/

/*---------------------------------------------------------------
-  FILE NAME     : Findinch.c
-  PROGRAMMER    : Terri A. Smith
-  DATE WRITTEN : January 1991
-  ADDRESS       : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE-  Finds the current unit grouping in the list of Is.
-       If it is not found program is exited.
-
-
-  MODIFICATION HISTORY-
-
- ---------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherlcl.h"

float find_inches(sizes)

   order_t sizes;

{
   int i, j;
   char match = 0;

   i = 0;
   while ((!match) && (i < num_list)) {
      match = 1;
      for (j=0; j<num_of_sizes; j++) {
         if (sizes[j] != list[i].sizes[j])
            match = 0;
         }
      ++i;
      }

   if (match)
      return(list[--i].inches);
   else {
      printf("\nCOULDNT FIND ");
      for (i=0; i<num_of_sizes; i++) {
      if (sizes[i] > 0)
         printf("%d %s ", sizes[i], order.ch_sizes[i]);
         }
      printf("\n");
      exit(0);
      }


}
```

```
/* ------------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/fives.c    January 1991
-- ------------------------------------------------------------------*/

/*------------------------------------------------------------------
-  FILE NAME    : fives.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : January 1991
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- Recursive procedure to gorup units in fives.
-
- ------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherlcl.h"

void fives(set_s, temp_secs, num_temp_secs)

   order_t set_s;
   section_t *temp_secs;
   int  *num_temp_secs;
{
   float inches;
   int j, i, k, l, m, n;
   order_t temp_order;
   float hold_inches1;
   float hold_inches2;
   int hold_temp_num;

   hold_temp_num = *num_temp_secs;
   hold_inches2 = total_inches;


   for (i=0; i<num_of_sizes; i++) {
      for (j=i+1; j<num_of_sizes; j++) {
         for (k=j+1; k<num_of_sizes; k++) {
            for (l=k+1; l<num_of_sizes; l++) {
               for (m=l+1; m<num_of_sizes; m++) {

                  for (n=0; n<num_of_sizes; n++)
                     temp_order[n] = 0;

                  if ((set_s[i] == 1) && (set_s[j] == 1) &&
                      (set_s[k] == 1) && (set_s[l] == 1) &&
                      (set_s[m] == 1)) {
                     temp_order[i] = 1;
                     temp_order[j] = 1;
                     temp_order[k] = 1;
                     temp_order[l] = 1;
                     temp_order[m] = 1;
                     inches = combine_inches(temp_order);
                     if (inches != (float) 0.0) {
                        for (n=0; n< num_of_sizes; n++)
                           temp_secs[*num_temp_secs].sizes[n] = 0;
```

```
58                         total_inches = total_inches + inches;
59                         temp_secs[*num_temp_secs].sizes[i] = 1;
60                         temp_secs[*num_temp_secs].sizes[j] = 1;
61                         temp_secs[*num_temp_secs].sizes[k] = 1;
62                         temp_secs[*num_temp_secs].sizes[l] = 1;
63                         temp_secs[*num_temp_secs].sizes[m] = 1;
64                         ++*num_temp_secs;
65                         }
66                      temp_order[i] = 0;
67                      temp_order[j] = 0;
68                      temp_order[k] = 0;
69                      temp_order[l] = 0;
70                      temp_order[m] = 0;
71
72                      for (n=0; n<num_of_sizes; n++) {
73                         if ((n != i) && (n != j) && (n != k) &&
74                             (n != l) && (n != m) && (set_s[n] == 1)) {
75                            temp_order[n] = 1;
76                            }
77                         }
78
79                      hold_inches1 = total_inches;
80                      ones(temp_order, temp_secs, num_temp_secs);
81                      check_inches(temp_secs, num_temp_secs);
82
83                      for (n=0; n<num_of_sizes; n++) {
84                         if ((n != i) && (n != j) && (n != k) &&
85                             (n != l) && (set_s[n] == 1)) {
86                            --*num_temp_secs;
87                            }
88                         }
89
90
91                      total_inches = hold_inches1;
92                      twos(temp_order, temp_secs, num_temp_secs);
93
94                      total_inches = hold_inches1;
95                      threes(temp_order, temp_secs, num_temp_secs);
96
97                      total_inches = hold_inches1;
98                      fours(temp_order, temp_secs, num_temp_secs);
99
100                     total_inches = hold_inches1;
101                     fives(temp_order, temp_secs, num_temp_secs);
102
103                     *num_temp_secs = hold_temp_num;
104                     total_inches = hold_inches2;
105
106                     }
107                  }
108               }
109            }
110         }
111      }
112   }
113
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/fours.c    January 1991
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : fours.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : January 1991
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- Recursive procedure to group units in fours
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherlcl.h"

void fours(set_s, temp_secs, num_temp_secs)

   order_t set_s;
   section_t *temp_secs;
   int  *num_temp_secs;
{
   float inches;
   int j, i, k, l, m;
   order_t temp_order;
   float hold_inches1;
   float hold_inches2;
   int hold_temp_num;

   hold_temp_num = *num_temp_secs;
   hold_inches2 = total_inches;


   for (i=0; i<num_of_sizes; i++) {
      for (j=i+1; j<num_of_sizes; j++) {
         for (k=j+1; k<num_of_sizes; k++) {
            for (l=k+1; l<num_of_sizes; l++) {

               for (m=0; m<num_of_sizes; m++)
                  temp_order[m] = 0;

               if ((set_s[i] == 1) && (set_s[j] == 1) &&
                   (set_s[k] == 1) && (set_s[l] == 1)) {
                  temp_order[i] = 1;
                  temp_order[j] = 1;
                  temp_order[k] = 1;
                  temp_order[l] = 1;
                  inches = combine_inches(temp_order);
                  if (inches != (float) 0.0) {
                     for (m=0; m< num_of_sizes; m++)
                        temp_secs[*num_temp_secs].sizes[m] = 0;
                     total_inches = total_inches + inches;
                     temp_secs[*num_temp_secs].sizes[i] = 1;
```

```
58                      temp_secs[*num_temp_secs].sizes[j] = 1;
59                      temp_secs[*num_temp_secs].sizes[k] = 1;
60                      temp_secs[*num_temp_secs].sizes[l] = 1;
61                      ++*num_temp_secs;
62                      }
63                   temp_order[i] = 0;
64                   temp_order[j] = 0;
65                   temp_order[k] = 0;
66                   temp_order[l] = 0;
67
68                   for (m=0; m<num_of_sizes; m++) {
69                      if ((m != i) && (m != j) && (m != k) &&
70                          (m != l) && (set_s[m] == 1)) {
71                         temp_order[m] = 1;
72                         }
73                      }
74
75                   hold_inches1 = total_inches;
76                   ones(temp_order, temp_secs, num_temp_secs);
77                   check_inches(temp_secs, num_temp_secs);
78
79                   for (m=0; m<num_of_sizes; m++) {
80                      if ((m != i) && (m != j) && (m != k) &&
81                          (m != l) && (set_s[m] == 1)) {
82                         --*num_temp_secs;
83                         }
84                      }
85
86
87                   total_inches = hold_inches1;
88                   twos(temp_order, temp_secs, num_temp_secs);
89
90                   total_inches = hold_inches1;
91                   threes(temp_order, temp_secs, num_temp_secs);
92
93                   total_inches = hold_inches1;
94                   fours(temp_order, temp_secs, num_temp_secs);
95
96                   *num_temp_secs = hold_temp_num;
97                   total_inches = hold_inches2;
98
99                   }
100                }
101             }
102          }
103       }
104    }
105
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/getparm.c    December 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Getparm.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : December 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- To read in the parameters from a file
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherlcl.h"

int get_parameters(ou_units, max_ply, max_sizes)

   int   *ou_units;
   int   *max_ply;
   int   *max_sizes;

{
   int i, j;
   FILE *fp = NULL;
   int quantity;
   int m;
   float temp;

   if ((fp =fopen("INPUT", "r")) == NULL) {
      printf("Cannot open input file  - getparm.c");
      exit(0);
      }


   /* set order and list values to -1 */
   for (i = 0; i < MAX_SIZES; i++) {
      order.number[i] = 0;
      order.ch_sizes[i][0] = 0;
      }

   for (i=0; i<MAX_LIST; i++) {
      list[i].inches = (float) 0.0;

      for (j = 0; j < MAX_SIZES; j++)
         list[i].sizes[j] = 0;
      }

    /* Input Units */
    fscanf(fp,"%d", ou_units);
    fscanf(fp,"%d", max_ply);
    fscanf(fp,"%d", max_sizes);

```

```
58
59      /* Input Order */
60      for (i = 0; i < MAX_SIZES; i++) {
61         fscanf(fp,"%hd", &order.number[i]);
62         if (order.number[i] == -1) {
63            order.number[i] = 0;
64            break;
65            }
66
67         fscanf(fp,"%s", order.ch_sizes[i]);
68         }
69
70      num_of_sizes = i;
71
72
73      /* Input List */
74      i=0;
75      while(1) {
76
77        fscanf(fp,"%d", &quantity);
78
79        if (quantity == -2)
80            break;
81
82        while (quantity != -1) {
83
84            fscanf(fp,"%d", &m);
85
86            if (m >= num_of_sizes) {
87               printf("ERROR in reading size variable - getparm.c");
88               exit(0);
89               }
90
91            list[i].sizes[m] = quantity;
92
93            fscanf(fp,"%d", &quantity);
94            }
95
96        fscanf(fp,"%f", &list[i].inches);
97
98        ++i;
99        }
100
101     fclose(fp);
102
103     return(i);
104  }
```

```
1   /* ---------------------------------------------------------------
2   -- $Header::   D:/cops/src/cherry/globals.h    January 1991
3   -- ---------------------------------------------------------------*/
4
5   /*---------------------------------------------------------------
6   -  FILE NAME    : Globals.h
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : January 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To declare all global variables
12  -
13  -
14  - ---------------------------------------------------------------*/
15  #include <stdio.h>
16  #include "cherdec.h"
17  #include "cherlcl.h"
18
19     ord_var_t order;
20
21     list_t    *list = NULL;
22
23     int     num_of_sizes;
24
25     int     num_list;
26
27     order_t temp_order;
28
29     section_t *sections = NULL;
30
31     int num_sections;
32
33     float total_inches = (float) 0.0;
34
35     float prev_inch = (float) 9999.0;
36
37     int num_hold_secs;
38
39     section_t *hold_secs;
40
41     int ply_height;
```

```
1   INCLUDES = cherdec.h cherlcl.h
2   LIBNAME = cherlib
3
4
5   OBJS = \
6           globals.obj \
7           getparm.obj \
8           findinch.obj \
9           combine.obj  \
10          ones.obj  \
11          chkinch.obj  \
12          cphold.obj \
13          clrtemp.obj \
14          twos.obj  \
15          threes.obj \
16          fours.obj  \
17          fives.obj  \
18          sixes.obj
19
20
21  .c.obj:
22          $(CC)
23          $(LIB)
24
25  globals.obj : globals.c $(INCLUDES)
26
27  getparm.obj : getparm.c $(INCLUDES)
28
29  findinch.obj : findinch.c $(INCLUDES)
30
31  combine.obj : combine.c $(INCLUDES)
32
33  ones.obj : ones.c $(INCLUDES)
34
35  twos.obj : twos.c $(INCLUDES)
36
37  threes.obj : threes.c $(INCLUDES)
38
39  fours.obj : fours.c $(INCLUDES)
40
41  fives.obj : fives.c $(INCLUDES)
42
43  sixes.obj : sixes.c $(INCLUDES)
44
45  chkinch.obj : chkinch.c $(INCLUDES)
46
47  cphold.obj : cphold.c $(INCLUDES)
48
49  clrtemp.obj : clrtemp.c $(INCLUDES)
50
51  cherry.obj : cherry.c $(INCLUDES)
52
53  cherry.exe : cherry.obj  $(OBJS)
54          cl cherry /link cherlib.lib
55
56
57  $(B)\cherry.exe : cherry.exe
```

```
58              $(CP)
59
60      $(I)\cherdec.h : cherdec.h
61              $(CP)
62
63
64
65
66
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/ones.c    January 1991
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Ones.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : January 1991
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- Groups units in ones and find inches
-
-
-  MODIFICATION HISTORY-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherlcl.h"

void ones(set_s, temp_secs, num_temp_secs)

   order_t set_s;
   section_t *temp_secs;
   int  *num_temp_secs;
{
   float inches;
   int j, i, m;

   j = *num_temp_secs;

   for (i=0; i<num_of_sizes; i++) {
      if (set_s[i] == 1) {

      for (m=0; m<num_of_sizes; m++)
         temp_secs[j].sizes[m] = 0;

         temp_secs[j].sizes[i] = 1;
         inches = find_inches(temp_secs[j].sizes);
         if (inches != (float) 0.0) {
            total_inches = total_inches + inches;
            ++j;
            }
         else
            temp_secs[j].sizes[i] = 0;
         }
      }

   *num_temp_secs = j;

}
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/sixes.c    January 1991
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : sixes.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : January 1991
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- Recursive procedure to group units in sixes.
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherlcl.h"

void sixes(set_s, temp_secs, num_temp_secs)

   order_t set_s;
   section_t *temp_secs;
   int  *num_temp_secs;
{
   float inches;
   int j, i, k, l, m, n, o;
   order_t temp_order;
   float hold_inches1;
   float hold_inches2;
   int hold_temp_num;

   hold_temp_num = *num_temp_secs;
   hold_inches2 = total_inches;


   for (i=0; i<num_of_sizes; i++) {
      for (j=i+1; j<num_of_sizes; j++) {
         for (k=j+1; k<num_of_sizes; k++) {
            for (l=k+1; l<num_of_sizes; l++) {
               for (m=l+1; m<num_of_sizes; m++) {
                  for (n=m+1; n<num_of_sizes; n++) {

                  for (o=0; o<num_of_sizes; o++)
                     temp_order[o] = 0;

                  if ((set_s[i] == 1) && (set_s[j] == 1) &&
                      (set_s[k] == 1) && (set_s[l] == 1) &&
                      (set_s[m] == 1) && (set_s[n] == 1)) {
                     temp_order[i] = 1;
                     temp_order[j] = 1;
                     temp_order[k] = 1;
                     temp_order[l] = 1;
                     temp_order[m] = 1;
                     temp_order[n] = 1;
                     inches = combine_inches(temp_order);
                     if (inches != (float) 0.0) {
```

```
58                      for (o=0; o< num_of_sizes; o++)
59                         temp_secs[*num_temp_secs].sizes[o] = 0;
60                      total_inches = total_inches + inches;
61                      temp_secs[*num_temp_secs].sizes[i] = 1;
62                      temp_secs[*num_temp_secs].sizes[j] = 1;
63                      temp_secs[*num_temp_secs].sizes[k] = 1;
64                      temp_secs[*num_temp_secs].sizes[l] = 1;
65                      temp_secs[*num_temp_secs].sizes[m] = 1;
66                      temp_secs[*num_temp_secs].sizes[n] = 1;
67                      ++*num_temp_secs;
68                      }
69                   temp_order[i] = 0;
70                   temp_order[j] = 0;
71                   temp_order[k] = 0;
72                   temp_order[l] = 0;
73                   temp_order[m] = 0;
74                   temp_order[n] = 0;
75
76                   for (o=0; o<num_of_sizes; o++) {
77                      if ((o != i) && (o != j) && (o != k) &&
78                          (o != l) && (o != m) && (o != n) && (set_s[o] == 1)) {
79                         temp_order[o] = 1;
80                         }
81                      }
82
83                   hold_inches1 = total_inches;
84                   ones(temp_order, temp_secs, num_temp_secs);
85                   check_inches(temp_secs, num_temp_secs);
86
87                   for (o=0; o<num_of_sizes; o++) {
88                      if ((o != i) && (o != j) && (o != k) &&
89                          (o != l) && (o !=n) && (set_s[o] == 1)) {
90                         --*num_temp_secs;
91                         }
92                      }
93
94
95                   total_inches = hold_inches1;
96                   twos(temp_order, temp_secs, num_temp_secs);
97
98                   total_inches = hold_inches1;
99                   threes(temp_order, temp_secs, num_temp_secs);
100
101                  total_inches = hold_inches1;
102                  fours(temp_order, temp_secs, num_temp_secs);
103
104                  total_inches = hold_inches1;
105                  fives(temp_order, temp_secs, num_temp_secs);
106
107                  total_inches = hold_inches1;
108                  sixes(temp_order, temp_secs, num_temp_secs);
109
110                  *num_temp_secs = hold_temp_num;
111                  total_inches = hold_inches2;
112
113                  }
114               }
```

```
115                     }
116                 }
117             }
118         }
119     }
120 }
121
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/cherry/threes.c    January 1991
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Threes.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : January 1991
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- Recursive procedure to group units in threes
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherlcl.h"

void threes(set_s, temp_secs, num_temp_secs)

   order_t set_s;
   section_t *temp_secs;
   int  *num_temp_secs;
{
   float inches;
   int j, i, k, l;
   order_t temp_order;
   float hold_inches1;
   float hold_inches2;
   int hold_temp_num;

   hold_temp_num = *num_temp_secs;
   hold_inches2 = total_inches;

   for (i=0; i<num_of_sizes; i++) {
      for (j=i+1; j<num_of_sizes; j++) {
         for (k=j+1; k<num_of_sizes; k++) {

            for (l=0; l<num_of_sizes; l++)
               temp_order[l] = 0;

            if ((set_s[i] == 1) && (set_s[j] == 1) && (set_s[k] == 1)) {
               temp_order[i] = 1;
               temp_order[j] = 1;
               temp_order[k] = 1;
               inches = combine_inches(temp_order);
               if (inches != (float) 0.0) {
                  for (l=0; l< num_of_sizes; l++)
                     temp_secs[*num_temp_secs].sizes[l] = 0;
                  total_inches = total_inches + inches;
                  temp_secs[*num_temp_secs].sizes[i] = 1;
                  temp_secs[*num_temp_secs].sizes[j] = 1;
                  temp_secs[*num_temp_secs].sizes[k] = 1;
                  ++*num_temp_secs;
                  )
```

```
58                temp_order[i] = 0;
59                temp_order[j] = 0;
60                temp_order[k] = 0;
61
62                for (l=0; l<num_of_sizes; l++) {
63                   if ((l != i) && (l != j) && (l != k) && (set_s[l] == 1)) {
64                      temp_order[l] = 1;
65                      }
66                   }
67
68                hold_inches1 = total_inches;
69                ones(temp_order, temp_secs, num_temp_secs);
70                check_inches(temp_secs, num_temp_secs);
71
72                for (l=0; l<num_of_sizes; l++) {
73                   if ((l != i) && (l != j) && (l != k) && (set_s[l] == 1)) {
74                      --*num_temp_secs;
75                      }
76                   }
77
78
79                total_inches = hold_inches1;
80                twos(temp_order, temp_secs, num_temp_secs);
81
82
83                total_inches = hold_inches1;
84
85   /*             for (l=0; l<num_of_sizes; l++) {
86                   if ((l != i) && (l != j) && (l != k) && (set_s[l] == 1)) {
87                      --*num_temp_secs;
88                      }
89                   }
90   */
91                total_inches = hold_inches1;
92                threes(temp_order, temp_secs, num_temp_secs);
93
94                *num_temp_secs = hold_temp_num;
95                total_inches = hold_inches2;
96
97                }
98             }
99          }
100       }
101   }
102
```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/cherry/twos.c    January 1991
3   -- ------------------------------------------------------------*/
4
5   /*------------------------------------------------------------------
6   -  FILE NAME    : Twos.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : January 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- Recursive procedure to group units in twos
12  -
13  -
14  - ------------------------------------------------------------*/
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include <string.h>
18  #include "cherdec.h"
19  #include "cherlcl.h"
20
21  void twos(set_s, temp_secs, num_temp_secs)
22
23     order_t set_s;
24     section_t *temp_secs;
25     int  *num_temp_secs;
26  {
27     float inches;
28     int j, i, k, m;
29     order_t temp_order;
30     float hold_inches1;
31     float hold_inches2;
32     int hold_temp_num;
33
34     hold_temp_num = *num_temp_secs;
35     hold_inches2 = total_inches;
36
37
38     for (i=0; i<num_of_sizes; i++) {
39        for (j=i+1; j<num_of_sizes; j++) {
40
41           for (k=0; k<num_of_sizes; k++)
42              temp_order[k] = 0;
43
44           if ((set_s[i] == 1) && (set_s[j] == 1)) {
45              temp_order[i] = 1;
46              temp_order[j] = 1;
47              inches = combine_inches(temp_order);
48              if (inches != (float) 0.0) {
49                 for(m=0; m<num_of_sizes; m++)
50                    temp_secs[*num_temp_secs].sizes[m] = 0;
51                 total_inches = total_inches + inches;
52                 temp_secs[*num_temp_secs].sizes[i] = 1;
53                 temp_secs[*num_temp_secs].sizes[j] = 1;
54                 ++*num_temp_secs;
55  /*               printf(" WITH TOTAL = %d\n", total_inches);  */
56                 }
57              temp_order[i] = 0;
```

```
58                temp_order[j] = 0;
59
60                for (k=0; k<num_of_sizes; k++) {
61                   if ((k != i) && (k != j) && (set_s[k] == 1)) {
62                      temp_order[k] = 1;
63                      }
64                   }
65
66                hold_inches1 = total_inches;
67                ones(temp_order, temp_secs, num_temp_secs);
68                check_inches(temp_secs, num_temp_secs);
69
70                for (k=0; k<num_of_sizes; k++) {
71                   if ((k != i) && (k != j) && (set_s[k] == 1)) {
72                      --*num_temp_secs;
73                      }
74                   }
75
76
77                total_inches = hold_inches1;
78                twos(temp_order, temp_secs, num_temp_secs);
79                *num_temp_secs = hold_temp_num;
80                total_inches = hold_inches2;
81
82                }
83             }
84          }
85
86    }
87
```

Appendix F Improvement Algorithm Source Code

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/case_ai.c    February 1991
3   -- ------------------------------------------------------------*/
4
5   /*------------------------------------------------------------
6   -  FILE NAME    : case_ai.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : February 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To determine savings if sizes in two sections are
12  -       the same
13  -
14  -
15  - ------------------------------------------------------------*/
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include "impdec.h"
19  #include "implcl.h"
20
21  float case_ai(sect1, portion, cut_cost)
22
23     section_t *sect1;
24     section_t *portion;
25     int     cut_cost;
26
27  {
28     int i;
29     int e = 0;
30     float savings;
31
32     for (i=0; i< num_of_sizes; i++) {
33           e = e + (order.perimeter[i] * sect1->sizes[i]);
34           e = e + (order.perimeter[i] * portion->sizes[i]);
35        }
36
37     savings = (float) cut_cost * e;
38
39     return(savings);
40
41  }
42
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/improv/case_aii.c    February 1991
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : case_aii.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : February 1991
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- To determine savings by lying sizes next to each
-       other instead of on top.
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include "impdec.h"
#include "implcl.h"

float case_aii(i, l, unit_cost)

   int i;
   int l;
   int  unit_cost;

{
   float savings = (float) 0.0;
   float sect1_inch;
   float sect2_inch;
   float sect3_inch;
   float sect4_inch;

   sect1_inch = find_inches(in_section[i].sizes);
   sect2_inch = find_inches(in_section[l].sizes);
   sect3_inch = find_inches(sect3.sizes);
   sect4_inch = find_inches(sect4.sizes);

   savings = unit_cost * in_section[i].ply_height * (sect1_inch + sect2_inch -
                                                     sect3_inch - sect4_inch);

   return(savings);

}
```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/combply.c    February 1990
3   -- ------------------------------------------------------------*/
4
5   /*------------------------------------------------------------
6   -  FILE NAME    : Combply.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : April 1990
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- This procedure combines sections which have the same sizes
12  -       and the ply heighth of the new section does not excede the max_ply
13  -
14  -
15  - ------------------------------------------------------------*/
16  #include <stdio.h>
17  #include <malloc.h>
18  #include <stdlib.h>
19  #include <memory.h>
20  #include "impdec.h"
21  #include "implcl.h"
22
23  void combine_ply(max_ply)
24     int max_ply;
25
26  {
27
28     int i,j,l,k;
29     char match;
30
31
32     for (i=0; i<num_in_sec; i++) {
33        for (j=i+1; j<num_in_sec; j++) {
34           match = 1;
35           for (k=0; k<num_of_sizes; k++) {
36             if (in_section[i].sizes[k] != in_section[j].sizes[k])
37                match = 0;
38             }
39           if ((match) &&
40               ((in_section[i].ply_height + in_section[j].ply_height) <= max_ply)) {
41               in_section[i].ply_height = in_section[i].ply_height + in_section[j].ply_height;
42               for (l=j; l<num_in_sec-1; l++)
43                  memcpy(&in_section[l], &in_section[l+1], sizeof(section_t));
44               --j;
45               --num_in_sec;
46               }
47           }
48       }
49
50     num_temp_sec = num_in_sec;
51
52     return;
53  }
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/improv/combsize.c    February 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Combsize.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : April 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- This procedure combines all sections in which the
-       the number of sizes in the new section does not excede
-       the max_sizes allowed per section
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <memory.h>
#include "impdec.h"
#include "implcl.h"


void combine_sizes(max_sizes)
   int max_sizes;

{
   int num_units;
   int i, j, l;

   for (i=0; i<num_in_sec; i++) {
      for (j=i+1; j<num_in_sec; j++) {
         num_units = 0;
         for (l=0; l<num_of_sizes; l++)
            num_units = num_units + in_section[i].sizes[l] +
                        in_section[j].sizes[l];

         if ((num_units <= max_sizes) &&
             (in_section[i].ply_height == in_section[j].ply_height)) {
             for (l=0; l<num_of_sizes; l++)
                in_section[i].sizes[l] = in_section[i].sizes[l] +
                                         in_section[j].sizes[l];

             for (l=j; l<num_in_sec-1; l++)
                memcpy(&in_section[l], &in_section[l+1], sizeof(section_t));

             --j;
             --num_in_sec;
             }
         }
     }

   num_temp_sec = num_in_sec;

   return;
}
```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/compswap.c    February 1991
3   -- ------------------------------------------------------------*/
4
5   /*-------------------------------------------------------------
6   -  FILE NAME    : compswap.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : February 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To determine which method to ue to compute
12  -       the savings
13  -
14  - -------------------------------------------------------------*/
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include "impdec.h"
18  #include "implcl.h"
19
20  float compute_swap_savings(i, l, cut_cost, unit_cost, max_sizes)
21
22     int i;
23     int l;
24     int  cut_cost;
25     int  unit_cost;
26     int max_sizes;
27
28  {
29     float savings;
30
31     if (in_section[i].ply_height == in_section[l].ply_height) {
32        savings = case_aii(i, l, unit_cost);
33        temp_save.type= 3;
34        temp_save.cand_ply_height = in_section[i].ply_height;
35        temp_save.org_ply_height = in_section[i].ply_height;
36        }
37
38     else {
39        temp_save.cand_ply_height = in_section[l].ply_height;
40        temp_save.org_ply_height = in_section[i].ply_height;
41        savings = case_aii(i, l, unit_cost);
42        temp_save.type= 4;
43        }
44
45     temp_save.savings = savings;
46
47     return(savings);
48
49  }
50
```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/compute.c    February 1991
3   -- ------------------------------------------------------------*/
4
5   /*------------------------------------------------------------
6   -  FILE NAME    : compute.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : February 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To determine which method to ue to compute
12  -       the savings
13  -
14  - ------------------------------------------------------------*/
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include "impdec.h"
18  #include "implcl.h"
19
20  float compute_savings(i, l, cut_cost, unit_cost, max_sizes)
21
22     int i;
23     int l;
24     int cut_cost;
25     int unit_cost;
26     int max_sizes;
27
28  {
29     int j;
30     float savings = (float) 0.0;
31     float save2;
32     char match = 1;
33     int num_units = 0;
34
35     for (j=0; j<num_of_sizes; j++) {
36        if (portion.sizes[j] != in_section[l].sizes[j])
37           match = 0;
38        num_units = num_units + sect4.sizes[j];
39        }
40
41
42     if (match) {   /* sizes in sections are the same */
43        if (num_units <= max_sizes) {
44           save2 = case_aii(i, l, unit_cost);
45
46           if (save2 > savings) {
47              temp_save.type= 2;
48              savings = save2;
49              if (in_section[i].ply_height != in_section[l].ply_height)
50                 temp_save.cand_ply_height = in_section[l].ply_height;
51              else  temp_save.cand_ply_height = in_section[i].ply_height;
52              temp_save.org_ply_height = in_section[i].ply_height;
53              }
54           }
55        }
56
57     else if ((in_section[i].ply_height == in_section[l].ply_height) && (num_units <= max_sizes)) {
```

```
58              savings = case_aii(i, l, unit_cost);
59              temp_save.type= 3;
60              temp_save.cand_ply_height = in_section[i].ply_height;
61              temp_save.org_ply_height = in_section[i].ply_height;
62              }
63
64        else if (num_units <= max_sizes) {
65           if (in_section[i].ply_height != in_section[l].ply_height)
66              temp_save.cand_ply_height = in_section[l].ply_height;
67           else  temp_save.cand_ply_height = in_section[i].ply_height;
68           temp_save.org_ply_height = in_section[i].ply_height;
69
70           savings = case_aii(i, l, unit_cost);
71           temp_save.type= 4;
72           }
73
74        temp_save.savings = savings;
75
76        return(savings);
77
78     }
79
```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/findinch.c    February 1991
3   -- ------------------------------------------------------------*/
4
5   /*------------------------------------------------------------
6   -  FILE NAME    : Findinch.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : February 1991
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To determine the number of inches in a section based
12  -         on the input list is
13  -
14  -
15  - ------------------------------------------------------------*/
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include <string.h>
19  #include "impdec.h"
20  #include "implcl.h"
21
22  float find_inches(sizes)
23
24     order_t sizes;
25
26  {
27     int i, j;
28     char match = 0;
29     char empty = 0;
30
31     i = 0;
32     while ((!match) && (i < num_list)) {
33        empty = 1;
34        match = 1;
35        for (j=0; j<num_of_sizes; j++) {
36           if (sizes[j] != list[i].sizes[j])
37              match = 0;
38           if (sizes[j] != 0)
39              empty = 0;
40           }
41        ++i;
42        }
43
44     if (empty)
45        return((float) 0.0);
46
47     if (match)
48        return(list[--i].inches);
49     else {
50        printf(" COULDNT FIND ");
51        for (i=0; i<num_of_sizes; i++) {
52        if (sizes[i] > 0)
53          printf("%d %s ", sizes[i], order.ch_sizes[i]);
54           }
55        printf("\n");
56        exit(0);
57        }
```

58
59

```
1   /* ---------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/getparm.c    February 1990
3   -- ---------------------------------------------------------------*/
4
5   /*---------------------------------------------------------------
6   -  FILE NAME    : Getparm.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : February 1990
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- To read the input parameters from a file
12  -
13  -
14  - ---------------------------------------------------------------*/
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include <string.h>
18  #include <malloc.h>
19  #include "impdec.h"
20  #include "implcl.h"
21
22  int get_parameters(ou_units, max_ply, max_sizes,
23                     cut_cost, unit_cost, old_ou_units)
24
25     int   *ou_units;
26     int   *max_ply;
27     int   *max_sizes;
28     int   *cut_cost;
29     int   *unit_cost;
30     int   *old_ou_units;
31
32  {
33     int i, j;
34     FILE *fp = NULL;
35     int quantity;
36     int m;
37
38     if ((fp =fopen("INPUT", "r")) == NULL) {
39        printf("Cannot open input file  - getparm.c");
40        exit(0);
41        }
42
43     /* set order and list values to -1 */
44     for (i = 0; i < MAX_SIZES; i++) {
45        order.number[i] = 0;
46        order.ch_sizes[i][0] = 0;
47        order.perimeter[i] = 0;
48        }
49
50     for (i=0; i<MAX_LIST; i++) {
51        list[i].inches = (float) 0.0;
52
53        for (j = 0; j < MAX_SIZES; j++)
54           list[i].sizes[j] = 0;
55        }
56
57
```

```
58      fscanf(fp,"%d", ou_units);
59      fscanf(fp,"%d", max_ply);
60      fscanf(fp,"%d", max_sizes);
61      fscanf(fp,"%d", cut_cost);
62      fscanf(fp,"%d", unit_cost);
63
64
65      /* Input Order */
66      for (i = 0; i < MAX_SIZES; i++) {
67         fscanf(fp,"%d", &order.number[i]);
68         if (order.number[i] == -1) {
69            order.number[i] == 0;
70            break;
71            }
72
73         fscanf(fp,"%d", &order.perimeter[i]);
74         fscanf(fp,"%s", order.ch_sizes[i]);
75         }
76
77      num_of_sizes = i;
78
79      fscanf(fp,"%d", &num_in_sec);
80
81      if ((in_section = (section_t *)malloc(num_in_sec * sizeof(section_t))) == NULL) {
82         printf("ALLOCATION ERROR - SECTIONS   getparm.c\n");
83         exit(0);
84         }
85
86      for (i=0; i<num_in_sec; i++) {
87         in_section[i].ply_height = 0;
88         for (m=0; m<num_of_sizes; m++) {
89            in_section[i].sizes[m] = 0;
90            }
91         }
92
93      i = 0;
94      /* Input Sections */
95      while(i < num_in_sec) {
96
97        fscanf(fp,"%d", &quantity);
98
99        while (quantity != -1) {
100
101           fscanf(fp,"%d", &m);
102
103           if (m >= num_of_sizes) {
104              printf("ERROR in reading size variable - getparm.c");
105              exit(0);
106              }
107
108           in_section[i].sizes[m] = quantity;
109
110           fscanf(fp,"%d", &quantity);
111           }
112        fscanf(fp, "%d", &in_section[i].ply_height);
113
114        ++i;
```

```
115         }
116
117         fscanf(fp,"%d", old_ou_units);
118
119
120       /* Input List */
121       i=0;
122       while(1) {
123
124         fscanf(fp,"%d", &quantity);
125
126         if (quantity == -2)
127             break;
128
129         while (quantity != -1) {
130
131             fscanf(fp,"%d", &m);
132
133             if (m >= num_of_sizes) {
134                printf("ERROR in reading size variable - getparm.c");
135                exit(0);
136                }
137
138             list[i].sizes[m] = quantity;
139
140             fscanf(fp,"%d", &quantity);
141             }
142
143         fscanf(fp,"%f", &list[i].inches);
144
145         ++i;
146         }
147
148       fclose(fp);
149
150       return(i);
151   }
152
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/improv/globals.h    February 1991
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Globals.h
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : February 1991
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- To declare all global variables
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include "impdec.h"
#include "implcl.h"

   ord_var_t order;

   list_t    *list = NULL;

   int     num_of_sizes;

   int     num_list;

   section_t *in_section = NULL;

   int     num_in_sec;

   int     num_temp_sec;

   section_t sect3;

   section_t sect4;

   section_t portion;

   savings_t temp_save;

   savings_t save;
```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/impdec.h    February 1990
3   -- ------------------------------------------------------------*/
4
5   /*------------------------------------------------------------
6   -  FILE NAME    : Impdec.h
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : February 1990
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332   (404) 894-8952
10  -
11  -  PURPOSE- To define all structures and procedures
12  -
13  -
14  - ------------------------------------------------------------*/
15  #ifndef IMPDEC_H
16  #define IMPDEC_H
17
18  #define MAX_LIST 1000
19  #define MAX_SIZES 25
20  #define MAX_SAVINGS 400
21
22
23  typedef int order_t[MAX_SIZES];
24
25  typedef char sizes_t[MAX_SIZES][10];
26
27  typedef struct {
28     order_t   number;
29     sizes_t   ch_sizes;
30     int   perimeter[MAX_SIZES];
31     } ord_var_t;
32
33  typedef struct {
34     order_t  sizes;
35     float   inches;
36     } list_t;
37
38  typedef struct {
39     order_t  sizes;
40     int      ply_height;
41     char     merged;
42     } section_t;
43
44  typedef struct {
45     int sect1;
46     int sect2;
47     int org_ply_height;
48     int cand_ply_height;
49     float savings;
50     int type;
51     order_t org;
52     order_t cand;
53     order_t in_sect1;
54     order_t in_sect2;
55     } savings_t;
56
57
```

```
58  int get_parameters(int *units, int *max_ply, int *max_sizes,
59                     int *cut_cost, int *unit_cost, int* old_ou_units);
60
61  float find_inches(order_t sizes);
62
63  float case_aii(int i, int j, int unit_cost);
64
65  float compute_savings(int i, int j, int cut_cost, int unit_cost, int max_sizes);
66
67  float compute_swap_savings(int i, int j, int cut_cost, int unit_cost, int max_sizes);
68
69  void combine_ply(int max_ply);
70
71  void combine_sizes(int max_sizes);
72
73  void transfer_forward(int i, int j, int l,
74                        int cut_cost, int unit_cost, int max_sizes, int max_ply);
75
76  void transfer_backwards(int i, int j, int l,
77                          int cut_cost, int unit_cost, int max_sizes, int max_ply);
78
79  void swap_forward(int i, int j, int l,
80                    int cut_cost, int unit_cost, int max_sizes, int max_ply);
81
82  void swap_backwards(int i, int j, int l,
83                      int cut_cost, int unit_cost, int max_sizes, int max_ply);
84
85  #endif
```

```
 1   /* ---------------------------------------------------------------
 2   -- $Header::   D:/cops/src/improv/Impdec.h    February 1990
 3   -- ---------------------------------------------------------------*/
 4
 5   /*---------------------------------------------------------------
 6   -  FILE NAME    : Implcl.h
 7   -  PROGRAMMER   : Terri A. Smith
 8   -  DATE WRITTEN : February 1990
 9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10   -
11   -  PURPOSE- To define all global variables
12   -
13   -
14   - ---------------------------------------------------------------*/
15   #ifndef IMPLCL_H
16   #define IMPLCL_H
17
18
19   extern ord_var_t order;
20   extern list_t *list;
21   extern int num_list;
22   extern int num_of_sizes;
23   extern section_t *in_section;
24   extern int num_in_sec;
25   extern int num_temp_sec;
26   extern section_t sect3;
27   extern section_t sect4;
28   extern section_t portion;
29   extern savings_t temp_save;
30   extern savings_t save;
31
32
33   #endif
```

```
1   /* ------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/improve.c    February 1990
3   -- ------------------------------------------------------------*/
4
5   /*------------------------------------------------------------
6   -  FILE NAME    : Improve.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : February 1990
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- The main program which controls flow of execution
12  -
13  -
14  - ------------------------------------------------------------*/
15  #include <stdio.h>
16  #include <malloc.h>
17  #include <memory.h>
18  #include <stdlib.h>
19  #include <string.h>
20  #include <time.h>
21  #include "impdec.h"
22  #include "implcl.h"
23
24  #define clock() time(NULL)
25
26  main(argv, argc)
27     int argv;
28     char *argc[];
29
30  {
31     /* Input Variables */
32     int   ou_units;          /* # of units over/under allowed */
33     int   old_ou_units;      /* # of units over/under allowed */
34     int   max_ply;              /* max ply height allowed       */
35     int   max_sizes;            /* # of sizes allowed / section  */
36     int   init_ply;             /* initial ply height           */
37     int   cut_cost;             /* cutting cost                 */
38     int   unit_cost;            /* unit cost                    */
39
40     /* Output Variables */
41     float tot_length;           /* the total amt of fabric needed*/
42     float tot_marker;           /* total fabric between markers  */
43     int   unit_dev;             /* deviation of units to cut from order */
44     int   unit_count;           /* units in all sections        */
45     int   order_count;          /* # of units in order          */
46     char  unit_string[10];      /* OVER or UNDER                */
47
48     int     i,j, k, l, r, s,m,n;  /* counters                   */
49     int     total_order = 0;      /* total # in order           */
50     float   inches;               /* inches in sections * ply   */
51     float   marker;               /* inches between markers     */
52     FILE    *fp;                  /* file pointer for output    */
53     order_t temp_order;           /* temp order                 */
54     clock_t start_time;           /* used for timing alg        */
55     clock_t end_time;             /* used for timing alg        */
56     double  total_time;           /* total execution time       */
57     char    mergers_possible = 1; /* while loop boolean         */
```

```
58      section_t temp_sec;          /* temporary section            */
59
60      start_time = clock();
61
62      if ((fp = fopen("OUTPUT", "w")) == NULL) {
63         printf("CANNOT OPEN OUTPUT FILE    savings.c\n");
64         exit(0);
65         }
66
67      if ((list = (list_t *)malloc(MAX_LIST * sizeof(list_t))) == NULL) {
68         printf("ALLOCATION ERROR FOR LIST    savings.c\n");
69         exit(0);
70         }
71
72      /*
73         Get parameters and print out first solution
74      */
75
76      num_list = get_parameters(&ou_units, &max_ply, &max_sizes,
77                                &cut_cost, &unit_cost, &old_ou_units);
78
79      tot_length = (float) 0.0;
80      tot_marker = (float) 0.0;
81      fprintf(fp, "MAX PLY = %d  MAX # OF UNITS PER SECTION = %d\n", max_ply, max_sizes);
82      fprintf(fp, "UNIT COST = %d cents  CUT COST = %d cents\n", unit_cost, cut_cost);
83      fprintf(fp, "ORDER\n");
84      for (i=0; i<num_of_sizes; i++) {
85         fprintf(fp, "%d SIZE %s\n", order.number[i], order.ch_sizes[i]);
86         }
87
88      fprintf(fp, "\n FIRST SOLUTION \n");
89      for (i=0; i<num_in_sec; i++) {
90         fprintf(fp, "SECTION %d HAS PLY = %d\n", i, in_section[i].ply_height);
91         for (j=0; j<num_of_sizes; j++) {
92            if (in_section[i].sizes[j] > 0) {
93               fprintf(fp, "       AND %d SIZE %s\n", in_section[i].sizes[j], order.ch_sizes[j]);
94               }
95            }
96         marker = find_inches(in_section[i].sizes);
97         inches = marker * in_section[i].ply_height;
98         fprintf(fp, "MARKER LENGTH = %7.2f  TOTAL LENGTH = %7.2f\n\n", marker, inches);
99         tot_length = tot_length + inches;
100        tot_marker = tot_marker + marker;
101        }
102     fprintf(fp, "TOTAL MARKER = %7.2f TOTAL LENGTH = %7.2f\n\n", tot_marker, tot_length);
103
104     /*
105        Initialize savings structures
106     */
107     for (i=0; i<num_of_sizes; i++) {
108        save.org[i] = 0;
109        save.cand[i] = 0;
110        temp_save.org[i] = 0;
111        temp_save.cand[i] = 0;
112        }
113
114     /*
```

```
115         combine any sections with a combination of sizes <= max_sizes
116      */
117
118      combine_sizes(max_sizes);
119      /*
120         Main Loop of program -
121         The loop begins by trying to place one sizes form one section
122         into another section.  Once all possible transferred are tested,
123         then the program tries swapping two sizes from two different
124         sections.  The loop begins with the first section.  The best
125         transfer or swap from this section is made and the next section
126         goes through the same tests etc.  Once all sections have been
127         exhausted then the same is repeated but backwards (starting
128         with the last section.  This whole process is repeated twice.
129      */
130      mergers_possible = 2;
131      while (mergers_possible > 0) {
132
133         /* combine any sections with same sizes by putting on
134            top of each other if it doesn't violate max ply height
135         */
136
137         combine_ply(max_ply);
138
139         /*
140            Attempt to reaasign one portion from original section
141            to a new section and calculate savings.  Merge only
142            the one with the greatest savings
143         */
144         for (i=0; i<num_in_sec; i++) {
145            for (j=0; j<num_of_sizes; j++) {
146               save.sect1 = -1;
147               save.sect2 = -1;
148               save.type = 0;
149               save.org_ply_height = 0;
150               save.cand_ply_height = 0;
151               save.savings = (float) 0.0;
152
153               for (m=0; m<num_of_sizes; m++)
154                  portion.sizes[m] = 0;
155               portion.ply_height = 0;
156
157               for (l=i+1; l<num_in_sec; l++) {
158
159                  transfer_forward(i, j, l, cut_cost, unit_cost, max_sizes, max_ply);
160
161                  swap_forward(i, j, l, cut_cost, unit_cost, max_sizes, max_ply);
162
163                }
164
165               /*
166                  Place portion into section.  If the two sections have
167                  different ply heights then the smallest ply height is
168                  given to both sections and the section with the larger
169                  ply height is added to the end of the section list with
170                  a ply height equal to larger ply minus the smaller ply
171               */
```

```
172             r = save.sect1;
173             s = save.sect2;
174             if (save.savings != (float) 0.0) {
175                printf("REPLACING PORTION %d %d\n", r, s);
176
177                in_section[r].ply_height = save.org_ply_height;
178                in_section[s].ply_height = save.cand_ply_height;
179
180                if (save.org_ply_height < save.cand_ply_height) {
181                   in_section[s].ply_height = save.org_ply_height;
182                   temp_sec.ply_height = save.cand_ply_height -
183                                         save.org_ply_height;
184
185                   for(m=0; m<num_of_sizes; m++)
186                      temp_sec.sizes[m] = save.in_sect2[m];
187
188                   if ((in_section = realloc(in_section, ((num_temp_sec + 1)
189                                        * sizeof(section_t)))) == NULL) {
190                      printf("REALLOCATION ERROR FOR INSECTION      improve2.c");
191                      exit(0);
192                      }
193
194                   memcpy(&in_section[num_temp_sec++], &temp_sec, sizeof(section_t));
195                   }
196
197                else if (save.org_ply_height > save.cand_ply_height) {
198                   in_section[r].ply_height = save.cand_ply_height;
199                   temp_sec.ply_height = save.org_ply_height -
200                                         save.cand_ply_height;
201
202                   for(m=0; m<num_of_sizes; m++)
203                      temp_sec.sizes[m] = save.in_sect1[m];
204
205                   if ((in_section = realloc(in_section, ((num_temp_sec + 1)
206                                        * sizeof(section_t)))) == NULL) {
207                      printf("REALLOCATION ERROR FOR INSECTION      improve2.c");
208                      exit(0);
209                      }
210
211                   memcpy(&in_section[num_temp_sec++], &temp_sec, sizeof(section_t));
212                   }
213
214                for(m=0; m<num_of_sizes; m++) {
215                   in_section[r].sizes[m] = save.org[m];
216                   in_section[s].sizes[m] = save.cand[m];
217                   }
218                }
219             } /* for j */
220          } /* for i */
221
222
223       /*
224          Perform the same sequence of events to transfer and swap
225          sizes but start at end of list and go backwards
226
227          Attempt to reassign one portion from original section
228          to a new section and calculate savings.  Merge only
```

```
229            the one with the greatest savings
230         */
231
232         num_in_sec = num_temp_sec;
233
234         for (i=num_in_sec-1; i>=0; i--) {
235            for (j=0; j<num_of_sizes; j++) {
236               save.sect1 = -1;
237               save.sect2 = -1;
238               save.type = 0;
239               save.org_ply_height = 0;
240               save.cand_ply_height = 0;
241               save.savings = (float) 0.0;
242
243               for (m=0; m<num_of_sizes; m++)
244                  portion.sizes[m] = 0;
245               portion.ply_height = 0;
246
247               for (l=i-1; l>=0; l--) {
248
249                  transfer_backwards(i, j, l, cut_cost, unit_cost, max_sizes, max_ply);
250
251                  swap_backwards(i, j, l, cut_cost, unit_cost, max_sizes, max_ply);
252                  }
253
254               r = save.sect1;
255               s = save.sect2;
256               if (save.savings != (float) 0.0) {
257                  printf("REPLACING PORTION %d %d\n", r, s);
258
259                  in_section[r].ply_height = save.org_ply_height;
260                  in_section[s].ply_height = save.cand_ply_height;
261
262                  if (save.org_ply_height < save.cand_ply_height) {
263                     in_section[s].ply_height = save.org_ply_height;
264                     temp_sec.ply_height = save.cand_ply_height -
265                                           save.org_ply_height;
266
267                     for(m=0; m<num_of_sizes; m++) {
268                        temp_sec.sizes[m] = save.in_sect2[m];
269                        }
270                     if ((in_section = realloc(in_section, ((num_temp_sec + 1)
271                                              * sizeof(section_t)))) == NULL) {
272                        printf("REALLOCATION ERROR FOR INSECTION      improve2.c");
273                        exit(0);
274                        }
275
276                     memcpy(&in_section[num_temp_sec++], &temp_sec, sizeof(section_t));
277                     }
278                  else if (save.org_ply_height > save.cand_ply_height) {
279                     in_section[r].ply_height = save.cand_ply_height;
280                     temp_sec.ply_height = save.org_ply_height -
281                                           save.cand_ply_height;
282
283                     for(m=0; m<num_of_sizes; m++) {
284                        temp_sec.sizes[m] = save.in_sect1[m];
285                        }
```

```
286
287                  if ((in_section = realloc(in_section, ((num_temp_sec + 1)
288                                          * sizeof(section_t)))) == NULL) {
289                     printf("REALLOCATION ERROR FOR INSECTION      improve2.c");
290                     exit(0);
291                     }
292                  memcpy(&in_section[num_temp_sec++], &temp_sec, sizeof(section_t));
293                  }
294
295               for(m=0; m<num_of_sizes; m++) {
296                  in_section[r].sizes[m] = save.org[m];
297                  in_section[s].sizes[m] = save.cand[m];
298                  }
299               }
300            }  /* for j */
301         } /* for i */
302
303
304      num_in_sec = num_temp_sec;
305      --mergers_possible;
306      }/* while */
307
308   /*
309      Remove sections that are empty
310   */
311
312   for (i=0; i<num_in_sec; i++) {
313      order_count = 0;
314      for (j=0; j<num_of_sizes; j++) {
315         order_count = order_count + in_section[i].sizes[j];
316         }
317      if (order_count == 0) {
318         for (j=i; j<num_in_sec-1; j++) {
319            memcpy(&in_section[j], &in_section[j+1], sizeof(section_t));
320            }
321         num_in_sec = num_in_sec - 1;
322         }
323      }
324
325   end_time = clock();
326   total_time = ((double) end_time - start_time) / CLK_TCK;
327
328   fprintf(fp, "\n\n*********************************************\n");
329   tot_length = (float) 0.0;
330   tot_marker = (float) 0.0;
331   unit_dev = 0;
332   order_count = 0;
333   unit_count = 0;
334
335   fprintf(fp, "THE # OF FINAL SECTIONS ARE : %d\n", num_in_sec);
336   for (i=0; i<num_in_sec; i++) {
337      fprintf(fp, "SECTION %d HAS PLY = %d\n", i, in_section[i].ply_height);
338      for (j=0; j<num_of_sizes; j++) {
339         if (in_section[i].sizes[j] > 0) {
340            fprintf(fp, "                    AND %d SIZE %s\n", in_section[i].sizes[j], order.ch_sizes[j])
341        unit_count = unit_count + (in_section[i].sizes[j] * in_section[i].ply_height);
342            }
```

```
343             )
344           marker = find_inches(in_section[i].sizes);
345           inches = marker * in_section[i].ply_height;
346           fprintf(fp, "MARKER LENGTH = %7.2f  TOTAL LENGTH = %7.2f\n\n", marker, inches);
347           tot_length = tot_length + inches;
348           tot_marker = tot_marker + marker;
349           )
350
351         for (j=0; j<num_of_sizes; j++)
352              order_count = order_count + order.number[j];
353
354         unit_dev = order_count - unit_count;
355         if (unit_dev > 0)
356             strcpy(unit_string, "UNDER");
357         else if (unit_dev == 0)
358             strcpy(unit_string, "\0");
359         else {
360             unit_dev = unit_dev * -1;
361             strcpy(unit_string, "OVER");
362             }
363
364         fprintf(fp, "TOTAL MARKER = %7.2f TOTAL LENGTH = %7.2f\n\n", tot_marker, tot_length);
365         fprintf(fp, "UNIT OVER/UNDER = %d %s", unit_dev, unit_string);
366         fprintf(fp, "\n\nTOTAL TIME = %f\n", total_time);
367
368
369       if (list != NULL)
370         free(list);
371
372
373       fclose(fp);
374
375       return(0);
376     }
```

```
1   INCLUDES = impdec.h implcl.h
2   LIBNAME = implib
3
4
5   OBJS = \
6           globals.obj \
7           getparm.obj \
8           findinch.obj \
9           case_aii.obj \
10          compute.obj  \
11          compswap.obj  \
12          combsize.obj  \
13          combply.obj  \
14          tranfrwd.obj \
15          swapfrwd.obj \
16          tranbkwd.obj \
17          swapbkwd.obj
18
19
20  .c.obj:
21          $(CC)
22          $(LIB)
23
24
25  globals.obj : globals.c $(INCLUDES)
26
27  getparm.obj : getparm.c $(INCLUDES)
28
29  findinch.obj : findinch.c $(INCLUDES)
30
31  case_aii.obj : case_aii.c $(INCLUDES)
32
33  compute.obj : compute.c $(INCLUDES)
34
35  compswap.obj : compswap.c $(INCLUDES)
36
37  combply.obj  : combply.c $(INCLUDES)
38
39  combsize.obj : combsize.c $(INCLUDES)
40
41  tranfrwd.obj : tranfrwd.c $(INCLUDES)
42
43  swapfrwd.obj : swapfrwd.c $(INCLUDES)
44
45  tranbkwd.obj : tranbkwd.c $(INCLUDES)
46
47  swapbkwd.obj : swapbkwd.c $(INCLUDES)
48
49  improve.obj : improve.c $(INCLUDES)
50
51  improve.exe : improve.obj  $(OBJS)
52          cl improve /link implib.lib
53
54
55  $(B)\improve.exe : improve.exe
56          $(CP)
57
```

```
58    $(I)\impdec.h : impdec.h
59            $(CP)
60
61
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/improv/swapbkwd.c    February 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Swapkwd.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : April 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- This procudure attempts to swap one size from one
-       with another size in a different section if feasible.  It
-       works from the end of the section list to the start.
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include "impdec.h"
#include "implcl.h"

void swap_backwards(i, j, l, cut_cost, unit_cost, max_sizes, max_ply)
   int i;
   int j;
   int l;
   int cut_cost;
   int unit_cost;
   int max_sizes;
   int max_ply;

{

   int   k, m, n;             /* counters                        */
   int   num_units;           /* num_units in one section        */


   for (n=0; n<num_of_sizes; n++) {
      if ((in_section[i].sizes[j] > 0) &&
          (in_section[l].sizes[n] > 0)) {

          for (m=0; m<num_of_sizes; m++) {
             sect3.sizes[m] = in_section[i].sizes[m];
             sect4.sizes[m] = in_section[l].sizes[m];
             }

          sect3.sizes[j] = sect3.sizes[j] - 1;
          sect3.sizes[n] = sect3.sizes[n] + 1;
          sect4.sizes[j] = sect4.sizes[j] + 1;
          sect4.sizes[n] = sect4.sizes[n] - 1;

          temp_save.sect1 = i;
          temp_save.sect2 = l;
          temp_save.type = 0;
          temp_save.org_ply_height = 0;
```

```
58              temp_save.cand_ply_height = 0;
59              temp_save.savings = (float) 0.0;
60
61              compute_swap_savings(i, l, cut_cost, unit_cost, max_sizes);
62
63              num_units = 0;
64              for (m=0; m<num_of_sizes; m++)
65                 num_units = num_units + sect4.sizes[m];
66
67              if ((temp_save.savings > save.savings) &&
68                  (num_units <= max_sizes) &&
69                  (temp_save.type > 0) &&
70                  (temp_save.cand_ply_height <= max_ply)) {
71                  memcpy(&save, &temp_save, sizeof(savings_t));
72
73                  for (m=0; m<num_of_sizes; m++) {
74                     if (temp_save.type != 1) {
75                        save.org[m] = sect3.sizes[m];
76                        save.cand[m] = sect4.sizes[m];
77                        save.in_sect1[m] = in_section[i].sizes[m];
78                        save.in_sect2[m] = in_section[l].sizes[m];
79                        }
80                     else
81                        save.cand[m] = in_section[i].sizes[m];
82                     } /* for m */
83                  } /* if */
84              }  /* if */
85          } /* for n */
86      return;
87  }
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/improv/swapfrwd.c    February 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Swapfrwd.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : April 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952

-  PURPOSE- This procudure attempts to swap one size from one
-       with another size in a different section if feasible.  It
-       works from the start of the section list to the end.
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include "impdec.h"
#include "implcl.h"



void swap_forward(i, j, l, cut_cost, unit_cost, max_sizes, max_ply)
   int i;
   int j;
   int l;
   int cut_cost;
   int unit_cost;
   int max_sizes;
   int max_ply;


{
   int   k, m, n;              /* counters                      */
   int num_units;

   for (n=0; n<num_of_sizes; n++) {
      if ((in_section[i].sizes[j] > 0) &&
          (in_section[l].sizes[n] > 0)) {

         for (m=0; m<num_of_sizes; m++) {
            sect3.sizes[m] = in_section[i].sizes[m];
            sect4.sizes[m] = in_section[l].sizes[m];
            }

         sect3.sizes[j] = sect3.sizes[j] - 1;
         sect3.sizes[n] = sect3.sizes[n] + 1;
         sect4.sizes[j] = sect4.sizes[j] + 1;
         sect4.sizes[n] = sect4.sizes[n] - 1;

         temp_save.sect1 = i;
         temp_save.sect2 = l;
         temp_save.type = 0;
         temp_save.org_ply_height = 0;
```

```
58            temp_save.cand_ply_height = 0;
59            temp_save.savings = (float) 0.0;
60
61            compute_swap_savings(i, l, cut_cost, unit_cost, max_sizes);
62
63            num_units = 0;
64            for (m=0; m<num_of_sizes; m++)
65               num_units = num_units + sect4.sizes[m];
66
67            if ((temp_save.savings > save.savings) &&
68                (num_units <= max_sizes) &&
69                (temp_save.type > 0) &&
70                (temp_save.cand_ply_height <= max_ply)) {
71                memcpy(&save, &temp_save, sizeof(savings_t));
72
73                for (m=0; m<num_of_sizes; m++) {
74                  if (temp_save.type != 1) {
75                     save.org[m] = sect3.sizes[m];
76                     save.cand[m] = sect4.sizes[m];
77                     save.in_sect1[m] = in_section[i].sizes[m];
78                     save.in_sect2[m] = in_section[l].sizes[m];
79                     }
80                  else
81                     save.cand[m] = in_section[i].sizes[m];
82                  } /* for m */
83                } /* if */
84            } /* if */
85         } /* for n */
86
87
88      return;
89   }
```

```
/* ------------------------------------------------------------
-- $Header::   D:/cops/src/improv/tranbkwd.c    February 1990
-- ------------------------------------------------------------*/

/*------------------------------------------------------------
-  FILE NAME    : Tranbkwd.c
-  PROGRAMMER   : Terri A. Smith
-  DATE WRITTEN : April 1990
-  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
-
-  PURPOSE- This procudure attempts to transfer one size from one
-       section into another section if feasible.  It works from
-       the end of the section list to the start.
-
-
- ------------------------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include "impdec.h"
#include "implcl.h"

void transfer_backwards(i, j, l, cut_cost, unit_cost, max_sizes, max_ply)
   int i;
   int j;
   int l;
   int cut_cost;
   int unit_cost;
   int max_sizes;
   int max_ply;

{

   int   k, m;              /* counters                      */
   int   num_units;         /* num_units in one section      */

   if (in_section[i].sizes[j] > 0)  {
      for (m=0; m<num_of_sizes; m++) {
         sect3.sizes[m] = in_section[i].sizes[m];
         sect4.sizes[m] = in_section[l].sizes[m];
         }

      sect3.sizes[j] = sect3.sizes[j] - 1;
      sect4.sizes[j] = sect4.sizes[j] + 1;
      portion.sizes[j] = 1;
      portion.ply_height = in_section[i].ply_height;

      temp_save.sect1 = i;
      temp_save.sect2 = l;
      temp_save.type = 0;
      temp_save.org_ply_height = 0;
      temp_save.cand_ply_height = 0;
      temp_save.savings = (float) 0.0;

      compute_savings(i, l, cut_cost, unit_cost, max_sizes);
```

```
58
59          num_units = 0;
60          for (m=0; m<num_of_sizes; m++)
61             num_units = num_units + sect4.sizes[m];
62
63          if ((temp_save.savings > save.savings) &&
64              (num_units <= max_sizes) &&
65              (temp_save.type > 0) &&
66              (temp_save.cand_ply_height <= max_ply)) {
67              memcpy(&save, &temp_save, sizeof(savings_t));
68
69              for (m=0; m<num_of_sizes; m++) {
70                 if (temp_save.type != 1) {
71                    save.org[m] = sect3.sizes[m];
72                    save.cand[m] = sect4.sizes[m];
73                    save.in_sect1[m] = in_section[i].sizes[m];
74                    save.in_sect2[m] = in_section[l].sizes[m];
75                    }
76                 else
77                    save.cand[m] = in_section[i].sizes[m];
78
79                 }
80              }
81          }
82
83       return;
84    }
```

```
1   /* ---------------------------------------------------------------
2   -- $Header::   D:/cops/src/improv/tranfrwd.c    February 1990
3   -- ---------------------------------------------------------------*/
4
5   /*---------------------------------------------------------------
6   -  FILE NAME    : tranfrwd.c
7   -  PROGRAMMER   : Terri A. Smith
8   -  DATE WRITTEN : April 1990
9   -  ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
10  -
11  -  PURPOSE- This procudure attempts to transfer one size from one
12  -       section into another section if feasible.  It works from
13  -       the beginning of the section list to the end.
14  -
15  - ---------------------------------------------------------------*/
16  #include <stdio.h>
17  #include <malloc.h>
18  #include <memory.h>
19  #include <stdlib.h>
20  #include <string.h>
21  #include <time.h>
22  #include "impdec.h"
23  #include "implcl.h"
24
25  void transfer_forward(i, j, l, cut_cost, unit_cost, max_sizes, max_ply)
26     int i;
27     int j;
28     int l;
29     int cut_cost;
30     int unit_cost;
31     int max_sizes;
32     int max_ply;
33
34  {
35
36     int   k, m;              /* counters                      */
37     int   num_units;         /* num_units in one section      */
38
39     if (in_section[i].sizes[j] > 0)  {
40
41        for (m=0; m<num_of_sizes; m++) {
42           sect3.sizes[m] = in_section[i].sizes[m];
43           sect4.sizes[m] = in_section[l].sizes[m];
44           }
45
46        sect3.sizes[j] = sect3.sizes[j] - 1;
47        sect4.sizes[j] = sect4.sizes[j] + 1;
48        portion.sizes[j] = 1;
49        portion.ply_height = in_section[i].ply_height;
50
51        temp_save.sect1 = i;
52        temp_save.sect2 = l;
53        temp_save.type = 0;
54        temp_save.org_ply_height = 0;
55        temp_save.cand_ply_height = 0;
56        temp_save.savings = (float) 0.0;
57
```

```
58          compute_savings(i, l, cut_cost, unit_cost, max_sizes);
59
60          num_units = 0;
61          for (m=0; m<num_of_sizes; m++)
62            num_units = num_units + sect4.sizes[m];
63
64          if ((temp_save.savings > save.savings) &&
65              (num_units <= max_sizes) &&
66              (temp_save.type > 0) &&
67              (temp_save.cand_ply_height <= max_ply)) {
68              memcpy(&save, &temp_save, sizeof(savings_t));
69
70              for (m=0; m<num_of_sizes; m++) {
71                 if (temp_save.type != 1) {
72                    save.org[m] = sect3.sizes[m];
73                    save.cand[m] = sect4.sizes[m];
74                    save.in_sect1[m] = in_section[i].sizes[m];
75                    save.in_sect2[m] = in_section[l].sizes[m];
76                    }
77                 else
78                    save.cand[m] = in_section[i].sizes[m];
79                 }/* for m */
80              }
81          } /* if */
82
83
84      return;
85    }
```